Algorithms for Topology-Aware Sensor Networks

Von der Carl-Friedrich-Gauß-Fakultät
der Technischen Universität Carolo-Wilhelmina zu Braunschweig

zur Erlangung des Grades

Doktor der Naturwissenschaften (Dr. rer. nat.)

genehmigte Dissertation

von
Dipl.-Math. Alexander Kröller,
geboren am 7. 10. 1974 in Berlin.

Eingereicht am: 8. 10. 2007
Mündliche Prüfung am: 23. 11. 2007
Referent: Prof. Dr. Sándor P. Fekete
Korreferenten: Prof. Dr. Roger Wattenhofer und Prof. Dr. Uwe T. Zimmermann

> The devices lived in the blood of the human race like viruses and passed from one person to the next during sex or any other exchange of bodily fluids; they were smart packets of data, just like the ones traversing the media network, and by mating with one another in the blood, they formed a vast system of communication, parallel to and probably linked with the dry Net of optical lines and copper wires. Like the dry Net, the wet Net could be used for doing computations—for running programs. And it was now clear that John Percival Hackworth was using it for exactly that, running some kind of vast distributed program of his own devising. He was designing something.

Neal Stephenson: “The Diamond Age: Or, a Young Lady’s Illustrated Primer”, 1995

This thesis results from my work in Prof. Fekete's group at TU Braunschweig. Many people assisted and guided me along the way, and I would like to express my thanks to them.

My PhD advisor, **Sándor Fekete**, gave me unbelievable amounts of freedom, while being there to assist me whenever I needed direction and guidance.

I worked in a collaborative research project, where I met the exceptionally open-minded **Dennis Pfisterer**, whose contributions, critics, and ideas where key to the success of both the research project and this PhD thesis.

I am grateful to **Ekkehard Köhler**, who brought up an interesting question and lots of ideas, which resulted in Chapter 5.

Several friends where there to proofread and improve all that nonsense I wrote, these are the crazy indian **Nitin Ahuja**, waste transport optimizer **Ronny "Google" Hansmann**, **Sebastian Orlowski** (willing & ready, yet unused), my personal WSN reality check **Dennis Pfisterer**, **Christiane Schmidt** (PhD ComicID 318), the only one in this list with a real job **Mathias Schulz**, and **Ines Spenke** (who performed the stunt of finishing the proofreading just in time before giving birth to her baby).

Yue Wang kindly shared her implementation of her boundary recognition algorithm [WGM06], saving Section 3.4 after I failed to implement it by myself.

Finally I would like to thank my wife **Katja** for all her support, for her willingness to live a chaotic and stressful life with me, and most of all for our son **Linus**.

Hail Eris! All hail Discordia!

Contents

Figures

Chapter 1

Introduction

In recent time, the study of wireless sensor networks (WSNs) has become a rapidly developing research area. A WSN is a network of small sensing devices, called *motes*, which can communicate over a wireless channel.

The development of motes was a straightforward step after different technologies became available at very low cost: Small sensors, embedded micro-controllers, transceivers for wireless communications, and small power sources. Simply integrating these technologies into a single embedded device opens up a completely new field of applications. This lead to the "Smart Dust" project [KKP99], where the vision of WSNs was initially proposed. Consider millions of tiny devices, at a size comparable to dust particles, and at a negligable cost of individual devices. One could deploy such a network simply by throwing the nodes from planes or vehicles, for example in hazardous areas or places that are inaccessible to humans. The nodes would then run algorithms for self-organization, build a stable networking infrastructure by themselves, and eventually start to survey the area. They would react to changes in their sensor readings and compare findings with nearby nodes to distinguish actual phenomena in the environment from local misreadings. They would find base stations that collect the network's status reports and alarm messages, so that information about the surveyed area would eventually be available to whomever deployed the network. For an example, see Figure 1.1, which sketches a sensor network deployed all over a forest, which measures temperature, watches for forest fires and relays an alarm to firefighters. Because sensor nodes are located close to the fire, it is detected much earlier than what is possible by classical observation methods, e.g., satellites, manned watch towers, and alike. The network continues to work and provides live reports during the firefight operation, and may even help locating or guiding lost people.

As visions go, it was soon discovered that this one was at least slightly overenthusiastic. This has many reasons, two big issues being:

- Tiny batteries hardly exist with sufficient capacity.
- Developing, programming, and installing software for millions of small devices is cumbersome and requires completely new approaches.

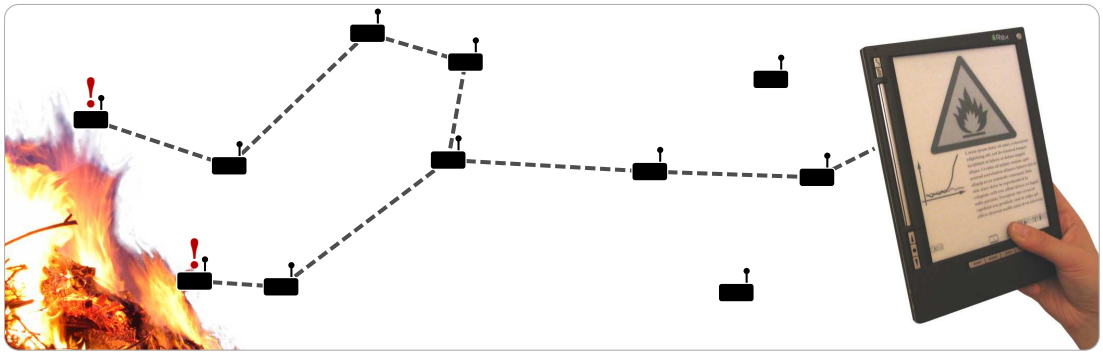


Figure 1.1: WSN observing a fire. *The two left-most nodes detect a fire in their vicinity. An alarm message is then routed over relay nodes to a base station.*

Over the last few years, research and development made great steps towards real-world sensor networks. Nowadays we see WSNs in practical applications in many different fields—from surveying glaciers to medical care. And while current devices are still large and quite costly, looking at the simpler yet similar RFID chips shows what is possible in the not-too-distant future. When RFID chips are already so small that they are indistinguishable from dust to the human eye, one can envision how small motes can become once they exist as highly integrated circuitry and are produced in large numbers.

With WSN research gaining popularity, it became evident that there were many algorithmic questions that needed to be addressed, especially because WSNs require a completely new kind of algorithms that did not exist previously. In the classic, centralized setting, an algorithm runs on a single processor, and has access to all problem data that exists at any point in time[1]. This makes it possible to use a unifying theory about algorithms, computability, problem complexity, and so on. A step towards WSNs is *parallel computing*, where a number of parallel processors jointly solve a problem. They still have shared access to some memory. *Distributed computing* takes this even further, assuming that each processor has only private memory, so nodes have to communicate to solve a problem together. With WSNs, several additional properties enter the stage:

- There is *geometry*—the network is placed in 2- or 3-dimensional space, and sensor values are only meaningful together with their location.
- Communication cannot happen between arbitrary nodes, as it is only possible when they are spatially close.
- Individual nodes may fail at any time. The loss of some nodes participating in

[1] this includes online algorithms, which merely cannot access *future* data.

a computation should not endanger the correct outcome of the overall algorithm.

- Each node is heavily limited, both in terms of processor speed and memory size. This comes from the design goal to have tiny motes at very low cost.
- Each node runs on a non-rechargeable tiny energy source, so heavy computation and communication is infeasible.

Therefore, there are many new problems associated with WSNs that require *distributed*, *geometry-aware*, and *energy-efficient* algorithms. This results in completely new (and exciting) paradigms for algorithmic research.

Our Vision: One of the basic algorithmic issues in WSNs is to let nodes know where they are. It is apparent that sensor readings are of little value, unless it is known where they were recorded. Furthermore, there is geometry in the network, so knowledge about it can be leveraged in many higher-level protocols and services, like multi-hop routing between nodes, tracking and addressing observed objects, generating a map showing hazard levels, and so on.

The first approaches to solve the localization problem were straightforward: Researchers added localization devices to the motes, for example GPS receivers. It quickly turned out that such receivers are quite costly and are not easily miniaturized. The second approach was to attach GPS to just a few nodes and let the other nodes "compute" their position from the known position of these so-called *anchor nodes*. Taking this to the next step, algorithms were developed that tried to assign positions to all nodes without using any external information at all. Unfortunately, it is now known that practically all variants of the localization problem are NP-hard, even in a classic, centralized setting. In distributed algorithms, there is the additional challenge that two nodes may get close positions by a localization algorithm, but are far away in reality—the localization is folded. Because they cannot communicate, they cannot detect this misplacement. They can only check that communication neighbors are indeed placed close to them, but this does not prevent folding at all. Using such position information can lead to all kinds of bad situations, e.g., a packet that arrives at a node seemingly close to the intended destination, yet far away in reality. Even worse, it may not be a packet but rather a firefighter, finally reaching the area where he incorrectly assumes the fire source to be.

There is one question that motivated most of our research for this thesis:

Why use Euclidean coordinates as localization information?

We tried to find an answer by looking at previous research and talking to engineers and computer scientists. We got many answers, but almost all of them boiled down to the following two points:

- Coordinates are naturally there—in simulations, visualizations, formulas, etc.
- It's been like that ever since motes were equipped with GPS devices.

Surprisingly, nobody could give us a reason why using coordinates as localization information would be better than alternative approaches. One frequently used argument was that coordinates allow to communicate an event's position to humans, e.g., marking the fire source on a map. This argument has two flaws: First, this presentation happens outside the network, through a full-fledged computer with a real processor and lots of memory. In this setting, it seems sub-optimal to put the computational burden of positioning onto the already crippled motes. Second, even if coordinates are required in-network one application, why should the network also use them for all other tasks that involve location?

We believe that Euclidean coordinates are a particularly poor choice of localization information. If a node knows it is at position 52°29'27"N 13°17'28"E, it can hardly use that information for anything useful. The numbers tells it nothing about the size of the network, its structure, the connectivity, the node's role in the network, or anything else. Even if it knows there is a base station waiting for reports at 52°26'39"N 13°21'31"E, it still has no clue how to relay a message towards it. This is a surprisingly hard task in coordinate-based WSNs, and a lot of research was necessary to provide actually working routing schemes. In the end, to get a picture of the whole network and its own position within it, a node needs to sample a lot of coordinates, which requires much communication, energy, and is counterproductive in the global goal of memory- and energy-efficiency.

Eventually, we came to the conclusion that the following question needs to be investigated further:

> *If we cannot (or don't want to) use global coordinates as localization information, how can we establish knowledge about the network's topology, and how can we use such knowledge to benefit the network's operation?*

This thesis describes most of the results stemming from this question. We focussed on scenarios where the network topology is complicated, with many holes and a complex shape of the area; this is where coordinate-based localization performs worst. We found a means to build *clusters*, that is, groups of nodes that claim to be a functional unit in the network, and construct a small geometric graph that precisely describes the network area. Every node knows to which cluster it belongs, and provable properties of the cluster decomposition make it possible to establish network services, say, routing with guaranteed delivery, at virtually no cost.

Organization of this Work: Chapter 2 describes the necessary fundamentals for the remainder of the thesis. This work uses bits and pieces from many disciplines,

so we focus on advanced topics, and assume the reader to be familiar with basic topics in mathematics and computer science.

Chapter 3 presents an algorithm to compute the boundary of a WSN without using coordinates; this is an important prerequisite for our clustering scheme. We also describe some competing algorithms both from ourselves and from the literature, and compare the different approaches by simulations.

Our topological clustering scheme is introduced in Chapter 4. We prove several beneficial properties of the clusters in the continuous case, and present algorithms for the discrete case. Furthermore, we describe how the clusters can be used for higher-level WSN services such as routing.

In Chapter 5, we introduce a novel algorithmic problem, the *Energy-Constrained Dynamic Flow* problem. While this was originally just another way to enhance clusters by computing characteristic properties, it became a solid piece of self-contained work.

Finally, Chapter 6 gives a short introduction into Shawn, a free WSN simulation software that we developed to run our algorithms in, and which is the only available software that serves the needs of algorithmic work in WSNs.

The work presented in this dissertation was not done in isolation. **Sándor Fekete** contributed to almost all aspects of this thesis. The chapter on dynamic flows (Chapter 5) is joint work with **Ekkehard Köhler** and **Alexander Hall**. The network simulator Shawn (see Chapter 6) came to live thanks to a fruitful collaboration with **Dennis Pfisterer** and contributions of many students, most notably **Tobias Baumgartner**.

Chapter 2

Basics

This chapter introduces fundamental concepts that are needed in later chapters. First there is an overview on today's motes and actual applications where they are used. Then we summarize basic results from selected topics in mathematics. We restrict this to just what we actually need in later chapters; it is not intended to be an complete introduction into the field. The third part of this chapter comprises models and basic properties related to sensor network theory, together with a short treatise on WSN localization.

2.1 Current Technology

In the eight years since the SmartDust project [KKP99] initially proposed the development of tiny motes, WSNs have left the field of pure academic research and are nowadays put into practice in a great variety of commercial applications.

2.1.1 Sensor Node Hardware

There are many ways to construct a mote. However, there is a basic set of components that are always present. They define the application space of WSNs, as well as the constraints under which theoretical research in this field has to operate.

Sensors: The purpose of most WSNs is sensing the environment, so there are sensors on the motes. They are the primary data source for the network. Actually, many WSN application are just that: Every node collects some data, which is then forwarded to a dedicated data sink, either directly or over relay nodes. A great variety of sensors is used in actual networks; they measure light, temperature, humidity, or acceleration, some motes even record audio. See the applications overview in Section 2.1.2 for more details.

Processor: By definition, a sensor node is an active computation device. It has a processor that is used for local computations on the data the node currently pos-

sesses, and to run communication protocols. Current motes feature anything from embedded micro-controllers running at 5 MHz, up to full-fledged CPUs, e.g., Intel's XScale series in the iMote 2, where the processor's speed can be adjusted between 13 and 400 MHz.

Memory: Motes have different kinds of memory. They often feature a small amount of RAM, usually between 5 and 256 kBytes as well as dedicated program storage. Some have flash-based memory to store large amounts of sensor data in, ranging up to 1 MByte in the iMote 2.

Communication: By definition, WSN motes can communicate with each other over a wireless channel. Currently, they often use standard communication technologies like WiFi/802.11, Bluetooth/802.15.1, or ZigBee/802.15.4. Some devices use alternatives such as sound or infrared light. Future devices will likely use different protocols, because none of the above are tailored for real low-energy communication with small protocol overhead.

Battery: The vision of sensor networks involves that each device has a small energy source that powers it for a while. When the battery is empty, the device simply dies, thereby removing itself from the network. The mote will not be recharged, it will just stay that way. Current motes usually use standard batteries. Because of the price, they are almost always collected after use, recharged, and reused.

Over the last years, several motes were developed for different kinds of requirements. Figures 2.1(a) to 2.1(d) show five different platforms that are currently in use. More devices, as well as detailed specifications, comparisons, and documentation can be found at the Sensor Network Museum[1]. It is noteworthy that none of these devices is highly integrated. All of them consist of off-the-shelf components. This adds to cost, size, and energy consumption: They usually cost around $50–100, meaning that not many organizations can afford a million-node network. They are large enough to strap two or three standard AA or AAA batteries to them, and they drain these batteries within a few days when no sleep-duty-cycle scheme is used.

Figure 2.1(d) shows a Spec mote. It is a highly integrated chip providing all circuitry that is needed for a sensor node. It was claimed that this chip can be produced in large numbers for less than $1. We are convinced that integrating single-chip motes, further miniaturization efforts, and time will yield much smaller and cheaper motes. Consider the μ-chip, see Figure 2.1(f). It is a complete RFID chip (shown here without antenna). It is much simpler than a sensor node: It is powered by the energy induced from a reading device communicating with it, and

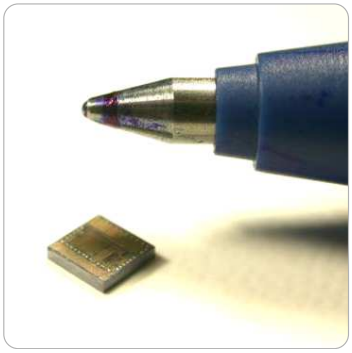
[1] `http://www.btnode.ethz.ch/Projects/SensorNetworkMuseum`

(a) MICA2 Mote, Crossbow Technology Inc.

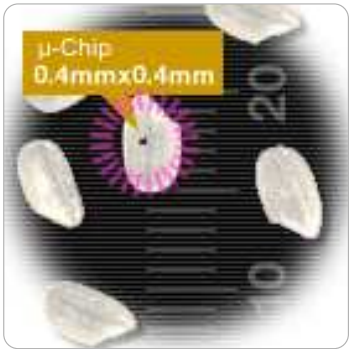
(b) Intel Mote (iMote) 2, Intel Corp.

(c) MSB430, ScatterWeb GmbH

(d) Tmote Mini, Moteiv Corp.



(e) Spec mote next to a ball-point pen, UC Berkeley



(f) RFID μ-Chip (black square) on a grain of rice, Hitachi Ltd.

Figure 2.1: Five sensor node platforms, and an RFID chip. *Image sources: (a,b) www.xbow.com; (c) cst.mi.fu-berlin.de/projects/ScatterWeb; (d) www.moteiv.com; (e) www.jlhlabs.com/jhill_cs/spec; (f) www.hitachi.co.jp/Prod/mu-chip .*

all it can do is send back a 128-bit ID number stored in ROM. It does perform this task in such a small chip that it is not perceivable by the human eye though.

2.1.2 Sensor Network Applications

One of the envisioned "killer applications" for sensor networks is a system that can be used for catastrophe recovery, e.g., in a forest fire, in a contaminated area, after an earthquake, and so on. The idea is that a sensor network is distributed in the area, so there is no time for pre-configuration or placement of nodes in carefully chosen positions. Once spread out, the network automatically organizes itself, establishes a mode of communication, obtains location knowledge and provides supportive data to the disaster response team. It monitors for hazards, locates and tracks helpless people, finds safe routes to them, and guides people back out.

This is still just a vision, but current technology proves that this will be possible in the near future. For now, there is a number of actually working sensor networks. We give an overview over a small number of applications.

Monitoring of Flora and Fauna: It is often not possible to monitor wild animals manually, because they are highly mobile, are disturbed by nearby humans, and because it is too expensive to pay full-time observers. Sensor networks were used successfully to monitor the habitats of birds [MPS+02, SMP+04], and they collect data about movements of wild zebras [JOW+02] in Kenya.

There are applications where plants are to be observed. In precision agriculture [Bag05], WSNs can report the micro-climates in a field and watch for conditions that may cause diseases. There are similar applications, stemming from ecological considerations, and recently the current global warming debate, for example WSNs to monitor a forest [BRY+04] or even the micro-climate around a single tree [TPS+05], and underwater networks for coral reefs [VKR+05].

Monitoring the Environment: Sensor networks are great for long-time observations of the environment. There is a network in Yosemite National Park that monitors meteorological and hydrologic processes [LCD03], and WSNs on volcanoes in Ecuador that watch for volcanic earthquakes [WASW06]. It is further possible to research the inner dynamic of glaciers by embedding a network deep in the ice [MPR+05]. Applications to increase safety include systems that detect landslides [AMTW05, STM+05].

Monitoring for Dangers: WSNs are successfully applied to increase safety. There is a smart tag system that can be attached to drums of hazardous chemicals. The tags make sure storage constraints are obeyed and communicate to drums in their vicinity to search for substances that must not be stored in the same place [KDD04]. It is also possible to measure the structural health of large constructions, e.g., houses, bridges, or ships, to raise an alarm prior to a collapse [XRC+04, CDB+04, KPC+06]. In heavy industry facilities, networks that sense sound and vibrations to detect failures in unmanned machinery are used [CKMS04, KAB+05].

People Surveillance: There are many ways how sensor networks can monitor individuals. There are medical applications where body-worn sensors that stream medical data to hospital servers while the patient is free to move around [SCL+05] or at home after treatment [KDD04]. Other applications include virtual trip-wires that detect, track and classify people in an area under surveillance [HCL+04].

Military Applications: There is a great amount of classified research on how sensor network can be utilized to help some people in killing some other people. Non-classified research includes networks to survey an area for moving enemy vehicles and troops [HKS+04], and an acoustic system to locate snipers [SMAL+04].

2.2 Mathematical Foundations

This section provides definitions and fundamental properties of the mathematical objects we use. The sensor networks we consider in this thesis are located in 2-dimensional space. Therefore, we restrict several definitions to $\mathbb{R}^2$. The purpose of this section is to present notation to avoid confusion; we assume that the reader is familiar with basic mathematical concepts.

We denote by $\mathbb{N}$ the set of positive integers and define $\mathbb{N}_0 := \mathbb{N} \cup \{0\}$. $\mathbb{Z}$ denotes the set of all integers, $\mathbb{Q}$ the set of all rational numbers, and $\mathbb{R}$ the set of all reals. The n-dimensional real vector space is denoted $\mathbb{R}^n$. For a vector $x \in \mathbb{R}^n$, $\|x\| := \|x\|_2 := (\sum_{i=1}^n x_i^2)^{1/2}$ denotes the Euclidean (or L_2) norm, $\|x\|_1 := \sum_{i=1}^n |x_i|$ the Manhattan (or L_1) norm, and $\|x\|_\infty := \max_{i=1,\dots,n} |x_i|$ the L_∞-norm. The Euclidean distance between two points $x, y \in \mathbb{R}^n$ is denoted by $\mathrm{d}(x,y) := \|x-y\|_2$.

For a point $x \in \mathbb{R}^2$ and a radius $r > 0$, $B_r(x) := \{y \in \mathbb{R}^2 : \mathrm{d}(x,y) \leqslant r\}$ is the *closed disk* with radius r around x. Furthermore, $B_r^\circ(x) := \{y \in \mathbb{R}^2 : \mathrm{d}(x,y) < r\}$ is the *open disk*, and $\partial B_r(x) := \{y \in \mathbb{R}^2 : \mathrm{d}(x,y) = r\}$ denotes the circle. For an arbitrary set $A \subseteq \mathbb{R}^2$, ∂A denotes the boundary of A.

We will also use some elementary probability theory. There, $\Pr[E]$ denotes the probability of an event E, and $\mathrm{E}[X]$ is the expectation of a random variable X.

2.2.1 Graphs

A *graph* $G = (V(G), E(G))$ (in short (V,E)) consists of a set V of nodes and a set E of edges. We restrict ourselves to undirected graphs without parallel edges and loops, that is, we assume $E \subseteq \{\{u,v\} : u,v \in V, u \neq v\}$. We use the simplified notation uv to denote edge $\{u,v\}$. We say that two nodes $u,v \in V$ are *adjacent*, if $uv \in E$. In this case, we say that u and uv are *incident*.

A *walk* in G is a sequence $W = (v_1, e_1, v_2, e_2, v_3, \dots, e_{k-1}, v_k)$ where $e_i = v_i v_{i+1}$ $\forall i = 1, \dots, k-1$. Because we do not allow parallel edges in our graphs, we also simply write $W = (v_1, \dots, v_k)$ and $W = (e_1, \dots, e_{k-1})$ (when v_1 and v_k are clear from context) to denote the same walk. Furthermore, for $v \in V$ ($e \in E$) we write $v \in W$ ($e \in W$) when the node (edge) appears in the walk. A walk that visits no node twice is called *path* or v_1-v_k-path. For two nodes $s, t \in V$, where $s \neq t$, we denote by $\mathscr{P}^{st}$ the set of all s-t-paths in G.

We denote by $\delta(v) := \{e \in E : v \in e\}$ the set of edges incident to a node $v \in V$.

The size $|\delta(v)|$ is called *degree* of v. The maximum degree in a graph is denoted by $\Delta(G) := \max_{v \in V} |\delta(v)|$. When the underlying graph is ambiguous, we clarify it using an index, as in $\delta_G(v)$.

The set of nodes adjacent to $v \in V$ is denoted by $N(v) := \{u \in V : uv \in E\}$. For a node set $U \subseteq V$, we use the notation $N(U) := \cup_{u \in U} N(u)$. Furthermore we define $N_0(v) := \{v\}$ and $N_k(v) := N(N_{k-1}(v)) \cup N_{k-1}(v)$ for $k \geqslant 1$. Again we use $N_k(U)$ analogous to $N(U)$. Trivially, $N_k(v)$ contains all nodes that can be reached from v by a walk with up to k edges. $N_k(v)$ is also called the *k-hop neighborhood* of v; this terminology stems from communication networks, where sending data over an a walk of k edges is called "using k hops". We also apply this notation to Δ and define $\Delta_k := \max\{|N_k(v)| : v \in V\}$.

2.2.2 Algorithms & Complexity

Here, we follow the definitions and notations of [GJ79, Pap94]. We formalize the notion of a mathematical problem as follows: A *problem* Π consists of *instances* $I \in \Pi$. The feasible answers to an instance are its solutions. An algorithm $\mathscr{A}$ is said to solve Π if it finds a feasible answer for every $I \in \Pi$. Two important classes of problems are

- *decision problems*, whose instances have the only feasible answers "yes" and "no", and
- *optimization problems*, whose instances consist of a *solution set* X and an *objective function* $f : X \to \mathbb{R}$. The answer to such a problem instance is either the value of $OPT(I) := \max_{x \in X} f(x)$, or "infeasible" if $X = \emptyset$, or "unbounded" if there is a divergent sequence $(x_n)_{n \in \mathbb{N}}$ in X with $x_n \to \infty$ $(n \to \infty)$. Problems where f has to be maximized are called *maximization* problems; cases with objective function $-f$ are called *minimization* problems.

To analyze the runtime of algorithms, the Landau symbols are quite useful. Let $f, g : \mathbb{N} \to \mathbb{R}$. We say that f *is* $O(g(n))$, in symbols $f(n) = O(g(n))$ or $f = O(g)$, if there exists a constant c such that $f(n) \leqslant cg(n)$ $\forall n \in \mathbb{N}$. The idea behind this definition is to say that f grows "in the order of g". We say f is *polynomially bounded*, if $f(n) = O(n^k)$ for some constant k. While the O-notation provides upper bounds, there is an analogous terminology for lower bounds: We say $f = \Omega(g)$, if $g = O(f)$, or equivalently, $f(n) \geqslant cg(n)$ $\forall n \in \mathbb{N}$ for some constant c. f has *exponential growth* if there exists some $k > 0$ such that $f(n) = \Omega(k^n)$. Finally, we say that $f = \Theta(g)$, if $f = O(g)$ and $g = O(f)$.

A problem instance $I \in \Pi$ has a size $\langle I \rangle$ which defines how many digits of an underlying alphabet are needed to encode I. The alphabet and encoding scheme are implicit parts of Π. Usually, a binary encoding is assumed, where the alphabet

is $\{0,1\}$. Here, numbers $p/q \in \mathbb{Q}$ are encoded in $O(\log|p| + \log|q|)$ bits. Another important scheme is *unary encoding*, where $\langle p/q \rangle = O(|p| + |q|)$.

We say a decision problem Π is in class P, if there exists an algorithm $\mathscr{A}$ solving Π such that a deterministic Turing machine running $\mathscr{A}$ always finishes within a number of operations that is polynomially bounded in $\langle I \rangle$. We say the Turing machine solves $\mathscr{A}$ *in polynomial time*. A problem is in NP, if there is a certificate for every "yes" instance of the problem that can be validated in polynomial time. Here, polynomial time refers to the problem instance, not the certificate. Note that these definitions are obviously incomplete, not having defined what a Turing machine actually is. An excellent overview with rigorous definitions can be found in [Pap94].

While P $\subseteq$ NP is easy to see, it is unknown whether the two classes actually differ. A problem Π is said to be NP-hard, if membership in P proves that P = NP. NP-hard problems that are in NP are called NP-complete problems. Such a problem is the famous SAT problem, see [GJ79] for many of them. Intuitively, NP-hardness means that a problem is at least as hard as every NP problem. Under the widely believed assumption P $\neq$ NP, no polynomial-time algorithm can exist for an NP-hard problem. Then, these problems cannot be solved efficiently. Some NP-hard problem are still hard if the implicit encoding scheme is changed from binary to unary. A trivial example is a problem that does not involve numbers. Such a problem with numbers in the input is *strong* NP*-hard*.

These problem classes only contain decision problems. There is a common sloppiness that allows to apply the same terms to optimization problems. Let $\max_{x \in X} f(x)$ be an optimization problem instance. We call this problem NP-hard, if the parameterized decision variant "*does there exist an* $x \in X$ *with* $f(x) \leqslant p$*?*" is NP-hard. An algorithm that produces an $x \in X$ with $f(x) \geqslant (1/c)\mathrm{OPT}$ is a *c-approximation*. An algorithm with parameter ε is a *polynomial-time approximation scheme* (PTAS), if it is a $(1+\varepsilon)$-approximation for every constant $\varepsilon > 0$, and its runtime is polynomial in the encoding of the problem. If it is also polynomial in $1/\varepsilon$, it is a *fully polynomial-time approximation scheme* (FPTAS).

2.2.3 Linear Programming

A *linear program* (LP) is a problem instance of the form

$$\max\{c^\mathsf{T} x : Ax \leqslant b\}, \tag{2.1}$$

where $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, and $b \in \mathbb{R}^m$. It can be seen as optimizing a linear objective function under m linear constraints $A_{i\cdot} x \leqslant b_i$, $i = 1, \ldots, m$. The n variables $x_1, \ldots, x_n$ are called *decision variables*. LPs are introduced in many different ways, e.g., with objective $\min c^\mathsf{T} x$, with nonnegative constraints $x \geqslant 0$ or additional equality constraints $A'x = b'$. Elementary algebra makes it possible to transform each of

these forms into every other one, hence we pick out the most simple one for the definition. With LP (2.1), another LP is associated:

$$\min\{y^\mathsf{T} b : y^\mathsf{T} A = c^\mathsf{T}\}. \tag{2.2}$$

It is called the *dual LP* of (2.1), which in turn is called the primal. Naturally, the dual's dual is the primal. The dual LP is of interest because of the strong duality theorem (see [Sch86] for an in-depth treatise), stating that (2.1) and (2.2) have the same optimal solution value, provided both are feasible.

The *separation problem* for an LP is the following: Given a point $z \in \mathbb{R}^n$, either state that it is feasible for (2.1) or provide a vector $a \in \mathbb{R}^n$ such that $a^\mathsf{T} x < a^\mathsf{T} z$ for all x that satisfy $Ax \leqslant b$.

Note that the separation problem can be solved by searching through the LP constraints for an index $i \in \{1, \ldots, m\}$ with $A_{i\cdot} z > b_i$. The dual separation problem is also frequently called the *pricing* problem, because the pricing step of the simplex algorithm for linear programming (again, see [Sch86]) does precisely this search.

The importance of separation lies in the equivalence of optimization and separation [GLS81]. It was shown that (2.1) can be solved in polynomial time, if there is an oracle that solves the separation problem in polynomial time. We will use this important fact excessively in Chapter 5, where we use LPs whose size is exponential in the given problem instance's input size, yet allowing for such an oracle for the dual separation problem.

A fractional *packing problem* is an LP of the type

$$\max\{c^\mathsf{T} x : Ax \leqslant b, x \geqslant 0\} \tag{2.3}$$

with an $m \times n$-matrix A, where all coefficients in A, b, and c are nonnegative. Its dual is the *covering LP*

$$\min\{y^\mathsf{T} b : y^\mathsf{T} A \geqslant c^\mathsf{T}, y \geqslant 0\}\,. \tag{2.4}$$

In [GK98], Garg and Könemann propose an algorithm that approximates problems of this type. In Chapter 5, we use a distributed variant of their algorithm, with an additional approximation step. Therefore, we repeat the extended algorithm here, see Algorithm 2.1. The main result about it is the following:

Theorem 2.1 (Garg, Könemann [GK98])**.** *Using an oracle that finds a Λ-approximation for the maximally violated constraint, the G&K algorithm computes a $\Lambda(1-\varepsilon)^{-2}$-approximation in $m\lceil \frac{1}{\varepsilon} \log_{1+\varepsilon} m \rceil$ iterations.*

Note that the original paper [GK98] only deals with optimal dual separation, i.e., $\Lambda = 1$. The extension for arbitrary $\Lambda > 1$ is straightforward and therefore skipped here.

Algorithm 2.1: Garg & Könemann with Λ-approximative separation

$\delta \leftarrow (1+\varepsilon)((1+\varepsilon)m)^{-1/\varepsilon}$, $x \leftarrow 0$, $y_j \leftarrow \delta / b_j \; \forall j = 1, \ldots, m$
while $y^\top b < 1$ **do**
  Find a $i^* \in \{1, \ldots, n\}$, where $c_{i^*} > 0$ and $(y^T A)_{i^*}/c_{i^*} \leqslant \Lambda (y^T A)_i / c_i$ for all $i = 1, \ldots, n : c_i > 0$
  $j^* \leftarrow \arg\min\{b_j / A_{i^* j} : \; j = 1, \ldots, m, \; A_{i^* j} \neq 0\}$
  $x_{i^*} = x_{i*} + b_{j^*} / A_{i^* j^*}$
  $y_j \leftarrow y_j (1+\varepsilon)(b_{j^*}/A_{i^* j^*})/(b_j / A_{i^* j})$ for all $j = 1, \ldots, m$ with $b_j \neq 0$
$x \leftarrow (1/\log_{1+\varepsilon}((1+\varepsilon)/\delta))x$

2.2.4 Network Flows

Static network flows are well-understood. See [AMO93] for an in-depth compendium on problems, problem variants, and algorithmic results. We assume G to be a directed graph, i.e., the edges vw and wv are treated as different elements of E, and it is possible that one of them is present in E while the other is not. To distinguish outgoing from incoming edges in $\delta(v)$, we split it into two sets, $\delta^+(v)$ and $\delta^-(v)$. The former contains the outgoing, the latter the incoming edges, i.e., for an edge $vw \in E$, $vw \in \delta^+(v)$ and $vw \in \delta^-(w)$ holds.

An edge $vw \in E$ can be used to send flow between v and w, as long as the flow does not exceed the capacity u_{vw}. We describe a flow using edge flow values, where f_{vw} is the flow amount on $vw \in E$. The *Maximum s-t-Flow* problem asks for a solution that sends as much flow as possible from a designated source $s \in V$ to a sink $t \in V$. It can be formulated as follows:

$$\max \sum_{sw \in \delta^+(s)} f_{sw} - \sum_{ws \in \delta^-(s)} f_{ws} \tag{2.5}$$

$$\text{s.t.} \sum_{vw \in \delta^+(v)} f_{vw} - \sum_{wv \in \delta^-(v)} f_{wv} = 0 \quad \forall v \in V \setminus \{s, t\} \tag{2.6}$$

$$0 \leqslant f_{vw} \leqslant u_{vw} \quad \forall vw \in E \, . \tag{2.7}$$

A closely related problem is the *Minimum-Cost Flow* problem, where the total amount

F is given, and a flow minimizing the edge costs $(c_e)_{e\in E}$ is sought:

$$\min \sum_{vw\in E} c_{vw} f_{vw} \tag{2.8}$$

$$\text{s.t.} \sum_{sw\in\delta^+(s)} f_{sw} - \sum_{ws\in\delta^-(s)} f_{ws} = F \tag{2.9}$$

$$\sum_{vw\in\delta^+(v)} f_{vw} - \sum_{wv\in\delta^-(v)} f_{wv} = 0 \quad \forall v \in V \setminus \{s,t\} \tag{2.10}$$

$$0 \leqslant f_{vw} \leqslant u_{vw} \qquad \forall vw \in E\,. \tag{2.11}$$

Both problems can be solved in polynomial time, even when integrality of the flow is required [AMO93].

Interesting variants arise when we add a notion of time: Assume that flow sent over an edge e does not appear at the destination immediately, but needs some time τ_e for the travel. Flows in such a setting are called *dynamic flows*.

Let $T \in \mathbb{N}$ be a *time horizon*. The *Maximum Dynamic Flow* problem asks for the maximum flow that can be sent from s to t within time T. A clarification is necessary to avoid "± 1 confusions": We follow the notation from [Hop95], where there are rounds $0,\dots,T$. Each link e can be used in rounds $0,\dots,T-\tau_e$. This reflects our wireless network scenario: If you have a time horizon of 1, you can send data over a link once. (There is an opposing model stemming from continuous flows, e.g., water flowing through a tube. There, you need a horizon of 2 for a unit-transit link: One round until the water reaches the destination, another until all the water is through.)

A simple way to model dynamic flows is using *time-expanded graphs*. Let $G = (V,E)$ be a graph, and let $T \in \mathbb{N}$. We construct a graph $G(T) = (V(T), E(T))$, where each node stands for a node of G at a particular time θ. We set

$$V(T) := \{v(\theta) : v \in V, \theta = 0,\dots,T\}, \tag{2.12}$$

$$E(T) := \{v(\theta)w(\theta+\tau_{vw}) : vw \in E, \theta = 0,\dots,T-\tau_{vw}\}\,. \tag{2.13}$$

Sometimes the application allows that flow pauses at certain nodes. In this case, $E(T)$ additionally contains the *holdover edges* $\{v(\theta)v(\theta+1) : v \in V, \theta = 0,\dots,T-1\}$, each having a transit time of one, and a capacity that models the storage capacity of the nodes.

The important point about time-expanded graphs is that a dynamic flow in the graph G is equivalent to a static flow in $G(T)$. Therefore, many dynamic flow problem can be solved by their static counterparts. It should be noted that T is exponential in $\langle T\rangle$, and thus $G(T)$ can have exponential size in the input size, turning polynomial static network algorithms into pseudo-polynomial algorithms for dynamic flows.

Network flows are usually presented and discussed for directed graphs, because the notation is much more straightforward then. In the problem we consider in Chapter 5, the edges are not directed. The capacity of the edges apply to the sum of both directions, because it models the total bandwidth of the channel between two nodes, independent of the direction.

However, this is not an issue. In the static case, the capacity constraints (2.7) and (2.11) need to be changed to read

$$f_{vw} + f_{wv} \leqslant u_{vw} \quad \forall vw = wv \in E \tag{2.14}$$
$$f_{vw} \geqslant 0 \quad \forall vw \in E \tag{2.15}$$

instead (note the slight ambiguity where f_{vw} is different from f_{wv}, but u_{vw} refers to the undirected edge). A formulation on the time-expanded graph uses

$$f_{v(\theta)w(\theta+\tau_{vw})} + f_{w(\theta)v(\theta+\tau_{vw})} \leqslant u_{vw} \quad \forall vw \in E, \theta = 0, \ldots, T - \tau_{vw} \tag{2.16}$$
$$f_{v(\theta)w(\theta+\tau_{vw})} \geqslant 0 \quad \forall vw \in E, \theta = 0, \ldots, T - \tau_{vw}\,. \tag{2.17}$$

2.2.5 Medial Axis

The medial axis $\mathbf{MA}(A)$ of a region A was originally proposed by Blum [Blu67]. It is a geometric graph that accurately describes the shape of A, as it is a strong deformation retract [SPW96], that is, there exists a homotopy $h : A \times [0,1] \to A$ where $h(a,0) = a\ \forall a \in A$, $h(a,1) \in \mathbf{MA}(A)\ \forall a \in A$, and $h(x,t) = x\ \forall x \in \mathbf{MA}(A), t \in [0,1]$. It can be interpreted as a kind of Voronoi diagram for an infinite set [FEdFC02].

We assume that ∂A consists of pairwise disjoint closed curves. Furthermore, each curve consists of a finite number of real analytic pieces. This guarantees the medial axis properties that we use [CCM97]. This does not restrict applicability of our results at all: In the network, we just consider a finite discrete sampling of A, so we could refine ∂A to adhere to all kinds of regularity requirements without affecting the network, e.g., as polygons, or sequences of straight lines and circular arcs.

We define by $\mathbf{D}(A)$ the poset of all closed disks contained in A, the partial order being set inclusion. The *core* of A, denoted $\mathbf{CORE}(A)$, consists of the maximal elements of $\mathbf{D}(A)$, i.e., all inclusion-wise maximal disks in A. The centers of these disks form the *medial axis* $\mathbf{MA}(A)$, and the center-radius pairs of them define the *medial axis transform* $\mathbf{MAT}(A)$. To be precise,

$$\mathbf{D}(A) := (\{B_r(x) \subseteq A : \ x \in A, r \geqslant 0\}, \subseteq), \tag{2.18}$$
$$\mathbf{CORE}(A) := \{B \in \mathbf{D}(A) : \ \forall B' \in \mathbf{D}(A)\, B' \supseteq B \Rightarrow B = B'\}, \tag{2.19}$$
$$\mathbf{MA}(A) := \{x : \ \exists B_r(x) \in \mathbf{CORE}(A)\}, \text{ and} \tag{2.20}$$
$$\mathbf{MAT}(A) := \{(x,r) : \ B_r(x) \in \mathbf{CORE}(A)\}. \tag{2.21}$$

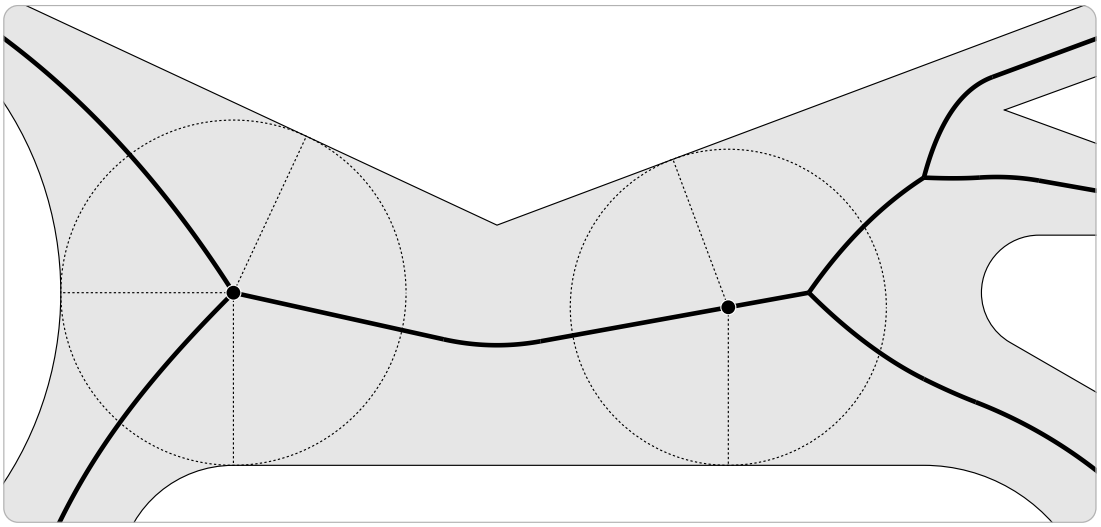


Figure 2.2: Medial axis of A. *The left marked point has three contact components, hence it is a bifurcation point.*

It is trivial to observe that for every $x \in \mathbf{MA}(A)$ there exists a unique $r \geqslant 0$ such that $(x, r) \in \mathbf{MAT}(A)$, and that $\mathbf{MA}(A)$ is the orthogonal projection of $\mathbf{MAT}(A)$ into the plane. Hence we will be sloppy with the distinction of these sets whenever it enhances readability. It is known that $\mathbf{MA}(A)$ is a connected and finite geometric graph [CCM97]. Informally, it consists of "medial lines" that run in the middle between boundaries, see Figure 2.2. For a $(x, r) \in \mathbf{MAT}(A)$, consider the set $C(x) := B_r(x) \cap \partial A$. The points in $C(x)$ are called *contact points* of x, and the connected components of $C(x)$ are called *contact components*. There are two kinds of components: Single points and circular arcs. Furthermore, the number of components is finite [CCM97]. A point on the medial axis is called *bifurcation point*, if it has at least three contact components. For an example, see the left medial axis point in Figure 2.2. Because bifurcation points coincidence with the vertices of the medial axis as a geometric graph, they are also called *medial vertices*. We denote by $V_3(A) \subseteq \mathbf{MAT}(A)$ the set of bifurcation points.

2.2.6 Distributed Computing

All of the aforementioned algorithms and results are based on the classic model of computation, where there is one processor, performing operations sequentially. A WSN is different, as there are many processors, each working independent of the others, trying to solve a global task together.

We adopt the *synchronous* model of Distributed Computing in this thesis. We present some fundamentals here. For more details, see the book [Pel00]. Other

great resources with a focus on WSNs are [WW07, ZG04].

In the synchronous model, there is a global clock that divides time into evenly sized *communication rounds*. In each round, every node can do the following:

1. Perform polynomial-time computations on the locally available data, and
2. send a message to each neighbor.

This is very close to the WSN setting, but it does not take the limited memory of sensor nodes into account. We assume that a node is able to store node IDs, which are of size $\Theta(\log n)$, and enough of them to get a picture of its k-hop neighborhood for constant k. Therefore, we allow each node to store data of size $O(\Delta_{O(1)} \log n)$.

The second of the above operations, sending messages, needs further explanation. Peleg [Pel00] proposes several models, of which we use two in this thesis:

- In the *LOCAL* model, each node can send unlimited amounts of data in each round.
- In the *CONGEST* model, messages are limited to a maximum size of $O(\log n)$.

The analysis of centralized algorithms hinges on their runtime function. Defining such a function for distributed systems is not trivial, so two useful complexity measures emerged:

- The *message complexity* of a distributed algorithm $\mathscr{A}$, denoted $\textsc{Message}(\mathscr{A})$, is the total number of messages being sent during the execution of $\mathscr{A}$. Note that it depends on the above models, as sending data with size s adds 1 in the *LOCAL* model, while it contributes $\lceil s/\log n \rceil$ in *CONGEST*.
- The *time complexity* $\textsc{Time}(\mathscr{A})$ is the number of rounds for the whole execution, measured from the first operation done in any node until the last node exits. Again, the complexity depends on the message model.

2.4 Network Models

As it is our goal to develop theoretical results for WSNs, we need clearly defined models and properties that describe the network.

Space: We assume that the nodes are located in a region $A \subset \mathbb{R}^2$ in the Euclidean plane. There is a mapping $p : V \to A$ that defines the actual positions of the nodes. We will frequently use p when we prove properties of this embedding. However, we do not assume that the nodes have access to p.

To ease notation, we will sometimes ignore the difference between a node v and its position $p(v)$. We will then treat v as a point in the plane, as in writing $\mathrm{d}(u,v)$ for the distance between two nodes, which would have to read $\mathrm{d}(p(u),p(v))$ to be correct.

The restriction to two-dimensional networks is not unrealistic. We consider huge networks only, spanning large geographic areas. Such networks may extend in the vertical direction, but only little compared to the horizontal size. When projecting the nodes onto $\mathbb{R}^2$, almost no information is lost.

Communication Cost: When a node v sends a signal to another node w, many factors need to be considered. The node sends with a certain *transmission power* P_v, but the signal strength degrades on the way. Many physical effects have an influence on how strong the signal at the receiver is. It is common to assume that w can successfully decode the signal if

$$\frac{P_v}{N + \mathrm{d}(v,w)^\alpha} \geqslant \beta \tag{2.22}$$

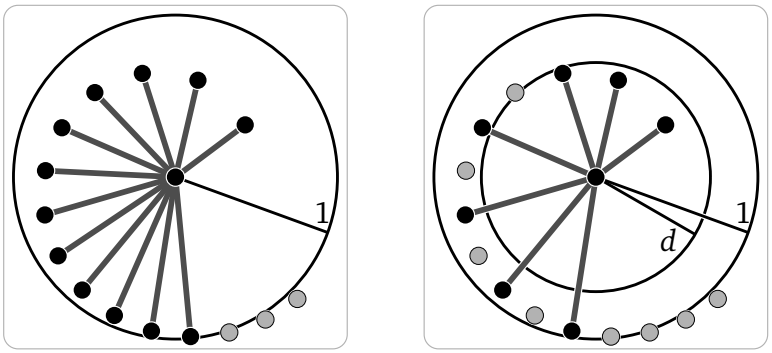
holds [WW07]. Here, α is a constant that depends on the environment and the medium through which the signal travels; it is usually assumed to be in the range $[2,5]$. The minimum signal strength that is sufficient for the receiving hardware to decipher the signal is denoted by β.

Communication Graph: By principle, a node cannot send with arbitrary power. Assuming $P_v \leqslant P_{\max}$ and solving inequality (2.22) for $\mathrm{d}(v,w)$, one comes to the conclusion that the area in which w can be is a perfect disk around v. If $P_{\max}$ is the same for all nodes, which is likely in a homogenous WSN where all nodes are of the same type, this disk has the same radius for all nodes. By scaling, we can assume this radius to be 1. This leads to the unit disk graph model:

Definition 2.2. *A graph $G = (V,E)$ is a* unit disk graph *(UDG), if there exists an embedding $p : V \to \mathbb{R}^2$ such that $\mathrm{d}(p(v),p(w)) \leqslant 1$ if and only if $vw \in E$, for all $v,w \in V$. In this case, p is called* UDG embedding *for G.*

See Figure 2.3(a) for a visualization. The communication range of real-world sensor nodes is almost never a perfect disk. The signal quality at a receiver depends on the material and shape of nearby objects, resulting in highly irregular communication ranges. To reflect this without losing too many combinatorial properties, there is the QUDG model.

Definition 2.3. *A graph $G = (V,E)$ is a d-*quasi unit disk graph *(d-QUDG) for $0 \leqslant d \leqslant 1$, if there exists an embedding $p : V \to \mathbb{R}^2$ such that*



(a) UDG (b) QUDG

Figure 2.3: Graph models. *The d-QUDG model is a generalization of UDGs that takes irregular communication ranges into acccount.*

(i) $\mathrm{d}(p(v), p(w)) \leqslant 1$ *for all edges* $vw \in E$*, and*
(ii) $\mathrm{d}(p(v), p(w)) > d$ *for all non-adjacent nodes* $v, w \in V : vw \notin E$*.*

In this case, p *is called* d-QUDG embedding *for* G*.*

See Figure 2.3(b) for a d-QUDG. The model was originally proposed in [BFN01], in the context of geometric routing. It was later thoroughly analyzed in [KWZ03]. The d-QUDG is a parameterized generalization of unit disk graphs, including both UDGs and general graphs:

- A 1-QUDG is a UDG.
- Every graph with n nodes is a $1/n$-QUDG.

The major reason why we introduce this model is that a $\sqrt{2}/2$-QUDG has the important *witness property*. It allows to deduce geometry from a graph without knowing its embedding p:

Lemma 2.4 (Witness Property)**.** *Let* u, v, w, x *be four different nodes in* V*, where* $uv \in E$ *and* $wx \in E$*. Assume the straight-line embeddings of* uv *and* wx *intersect. Then at least one of the edges in* $F := \{uw, ux, vw, vx\}$ *is also in* E*.*

Proof. We assume $p(u) \neq p(v)$; otherwise the lemma is trivial. See Figure 2.4. Let $a := \|p(u) - p(v)\|_2 \leqslant 1$. Consider two circles of common radius d with their centers at $p(u)$, resp. $p(v)$. The distance between the two intersection points of these circles is $h := 2\sqrt{d^2 - \frac{1}{4}a^2} \geqslant 1$. If F and E were disjoint, $p(w)$ and $p(x)$ both had to be outside the two circles. Because of the intersecting edge embeddings, $\|p(w) - p(x)\|_2 > h \geqslant 1$, which would contradict $wx \in E$. □

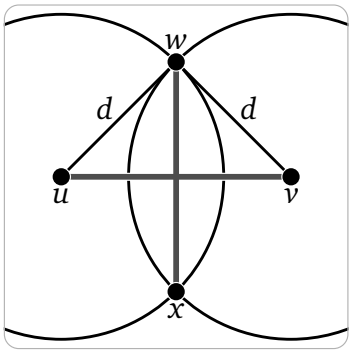


Figure 2.4: Witness property. *In a $\sqrt{2}/2$-QUDG, communication links cannot cross without being witnessed.*

This lemma is central to all our work on boundary recognition and clustering (Chapters 3 and 4). Therefore, we assume throughout this thesis that communication graph of the sensor network is a $\sqrt{2}/2$-QUDG.

It should be noted that the witness property cannot be generalized to higher dimension, where simplices take the role of edges:

Lemma 2.5. *In $\mathbb{R}^n$, $n \geqslant 3$, there is no Witness Property.*

Proof. Consider the $n-1$-dim Simplex $D := \frac{1}{2}\sqrt{2}\Delta^n = \text{conv}\{\frac{1}{2}\sqrt{2}e_1, \ldots, \frac{1}{2}\sqrt{2}e_n\} \subset \mathbb{R}^n$. The distance between any two vertices of it equals 1. Next let $p := \frac{1}{\|\mathbb{1}\|}\mathbb{1}$ and $\lambda = 1/\sqrt{2}$. Then $\lambda p \in D$. Construct two points $q_1 = (\lambda - \frac{1}{2})p$ and $q_2 = (\lambda + \frac{1}{2})p$. Now $\mathrm{d}(q_1, q_2) = 1$. But

$$\mathrm{d}(\tfrac{1}{2}\sqrt{2}e_i, q_j) = (n-1)\frac{1}{4n^2}(\sqrt{2}+\sqrt{n})^2 + (\frac{1}{2n}(\sqrt{2}+\sqrt{n}) - \frac{1}{\sqrt{2}})^2 > 1$$

for any $i \in \{1, \ldots, n\}$, $j \in \{1, 2\}$, and $n \geqslant 3$. □

2.5 Localization Woes

Much work on localization is rooted in a simple geometric fact: If the distances between every two points in a set are known, the point locations are unique (up to translation and rotation), and computing these positions is easy (in the sense of both computational complexity and mathematical effort). The basic technique of computing the position of a point when the distances to k reference points are known is called *multilateration*; the special case $k = 3$ is called *trilateration*.

The basic principles behind this can be carried over to situations where only a few distances are known, corresponding to the edges in the graph. There are graphs where a greedy sequence of lateration steps is sufficient to obtain a globally feasible

localization [EGW+04]. Hence, there is an uncountable number of algorithms that attempt to solve the localization problem, based on the basic connectivity information, enriched with different forms of distance estimation [ZG04, WW07, LR03].

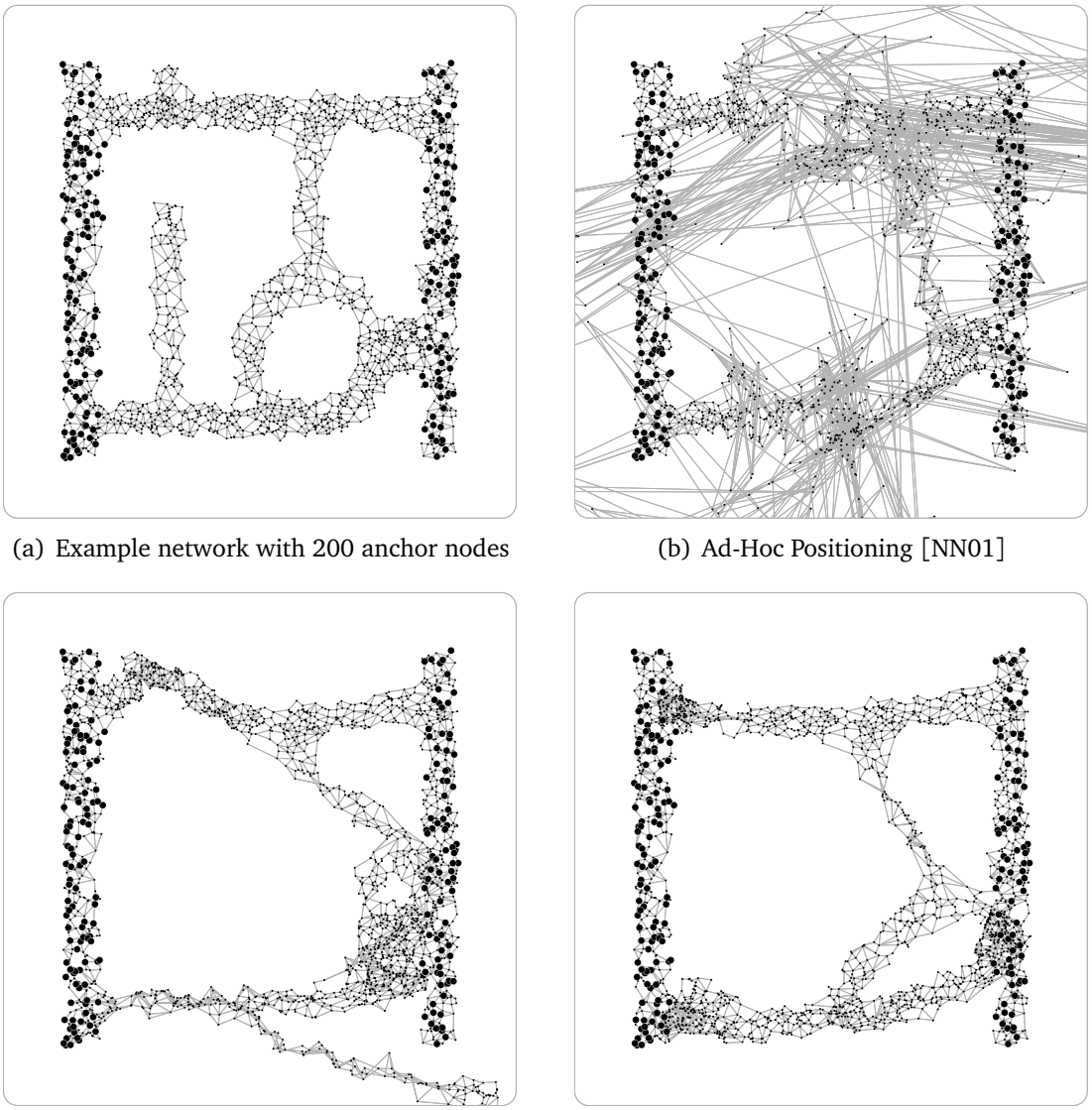
On the other hand, localization is a hard problem. This comes from the large number of negative results covering virtually all cases that are relevant in practice. All of the following problems are NP-hard:

1. Given a graph, decide whether it is a UDG [BK98].
2. Given a UDG, find a $(\sqrt{2}/3 + \varepsilon_n)$-QUDG embedding [KMW04] (here, $\varepsilon_n \to 0$ as $n \to \infty$).
3. Given a d-QUDG, find a d'-QUDG embedding, where $d' \geqslant \max\{\sqrt{2}/2, (\sqrt{2}/3 + \varepsilon_n)d\}$ [KMW04] (again, $\varepsilon_n \to 0$ as $n \to \infty$).
4. Given a UDG together with perfect edge lengths $\mathrm{d}(u, v)$ $\forall uv \in E$, find a UDG embedding that obeys these distances [AGY04].
5. Given a UDG together with perfect angle measurements, i.e., the angles between adjacent edges, find a $\sqrt{2}/2$-QUDG embedding [BGJ05a].
6. For every $\varepsilon > 0$: Given a UDG together with distances and angle measurements that have an absolute error of up to ε, find an embedding that respects these measurements within the given error [BGMS06].

So there obviously is a large gap between these hardness results and the existence of many localization algorithms. As it turns out, most algorithms are heuristics with a remarkable amount of engineering ingenuity, but no approximation guarantees. The only theoretically sound algorithm of which we are aware is a centralized and randomized algorithm that takes a UDG and produces an embedding as an $\Omega(1/(\log^{2.5} n\sqrt{\log\log n}))$-QUDG, with high probability [MOWW04].

Many localization algorithms attempt to simplify the problem using *anchors*. These are nodes that know their position using some external means, like a GPS device. The hope is that nodes can sidestep the inherent difficulties if they have error-free positions for some nodes just a few hops away.

The localization algorithm still needs to find consistent positions for all non-anchor nodes. We analyzed prominent algorithms by running simulations [FKB+05]. Here, the scenario models a portion of the network that lies in-between a number of anchors. As any network with anchors decomposes into a number of cells where all anchors are on the boundary, we see no benefit in looking at the complete network, but rather focus on a single cell. See Figure 2.5(a) for the network; it has 200 anchor nodes at the sides and 2000 nodes whose positions are to be calculated. Only a subset of the edges is drawn, to maintain readability. The network is a UDG, and



(a) Example network with 200 anchor nodes

(b) Ad-Hoc Positioning [NN01]

(c) Robust Positioning [SRL02]

(d) N-Hop Multilateration [SPS02]

Figure 2.5: Localization algorithms. *The marked nodes know their exact positions. Three algorithms computed the remaining positions of network (a). Their output is shown in (b)–(d).*

distance measures were available to the nodes. To simulate measurement errors, the edge lengths were disturbed with 1% error. This is tiny, compared with real-world applications. We evaluated three algorithms:

- In *Ad-Hoc Positioning* [NN01], nodes use a repeated triangulation procedure to guess their distance to anchors, then they position themselves by a single multilateration computation. See Figure 2.5(b). While this produces perfect

results, given error-free distances and a sufficiently many nodes in general position, it is clearly visible that this works only within a few hops from the anchors.

- *Robust Positioning* [SRL02] works without measuring distances. First, the anchors determine the average length of edges. The other nodes then use this value to estimate their distance to some anchors, and use multilateration to obtain an initial position. Finally, they repeatedly improve this position by multilateration to their direct neighbors. This final iteration ensures that nodes that are close in reality are also close in the computed position. Alas, it does not prevent folding at all. See Figure 2.5(c).

- *N-Hop Multilateration* [SPS02] is very similar to the previous one. It focusses on the final iteration, and uses a simple procedure for the initial positions: Each node estimates its distance to some anchor by summing distance estimates over shortest paths, and then finds a position that does not exceed these distances in the L_∞ metric. Again, the multilateration procedure ensure a seemingly nice embedding. The output still suffers from folding, see Figure 2.5(d).

Summarizing this, it is obvious that the localization problem is hard. Many problem variants are NP-hard, that is, difficult to solve even in a centralized setting. There is just one approximation algorithm, with a huge gap. Furthermore, distributed algorithms fail to solve the problem in complex topologies.

One could say that the localization problem is still mostly open, that there are many exciting questions to be answered, and hence much further research is needed. One could also say that this problem is obviously hard in theory and practice, so there is a need for alternate kinds of location knowledge and for methods by which WSNs can operate without classic localization. This is the key point of this thesis, and the following chapters will present such an alternative.

Chapter 3

Boundaries

In this chapter, we discuss how to detect the boundary of the sensor network. As the boundary is a geometrically defined property, but the nodes have no location information available, this is a challenging task. A working algorithm for this problem is a necessary precondition for the segmentation scheme in Chapter 4.

3.1 Problem Statement

The biggest issue with boundary recognition is that there is no precise definition of the actual problem. Even if the real embedding p of the nodes is known, it is still not clear what the "boundary" of the network is. One possibility for such a definition is the following: Assume the nodes are distributed over the area $A \subset \mathbb{R}^2$. Now define the area's boundary ∂A as the boundary of the network. This raises the following question: If the network is not aware of the global coordinate system, how could it describe ∂A? So, instead of seeking ∂A itself, one tries to identify all nodes that are "close to the boundary", i.e., by letting the network construct the set $B(A) := \{v \in V : \mathrm{d}(v, \partial A) \leqslant \delta\}$ for some parameter $\delta > 0$. If the network is distributed over A, it is also distributed over any $A' \supset A$, and whatever $B(A)$ is, $B(A + B_{2\delta}(0))$ is empty. Thus, this definition is nonsensical unless we require the network to "span" all of A, e.g., following a random distribution whose density function's support is A, or by requiring that the nodes are dense all over A. We present some algorithmic results for such a model in Section 3.2, where we consider a network with a uniform distribution on A. Alternately, one could define A based on the network. A common approach is to set $A := \bigcup_{v \in V} B_s(v)$, where s is the sensor radius of the nodes. In this case, A is the *coverage area* of the sensor network.

The approach of defining the boundary of the network with the help of ∂A is unsatisfactory—our goal is to be able to describe the boundary for as much networks as possible, not just those that behave nicely on A. An important aspect of boundary detection algorithms is how low the network density can become.

In the end, there is no universal definition of the boundary. Every algorithm is tailored for a certain definition, and is likely to be the only algorithm for it. In

Section 3.3 we present an algorithm that essentially defines the boundary to be whatever it produces, which is not uncommon in this area.

As a consequence, we compare algorithms using simulations. We use a particularly hard network, featuring a complex non-convex maze and a zone of fading density where defining a "correct" boundary line is hard even for humans. We present several relevant competitors to our algorithm and discuss strengths and weaknesses of each.

3.1.1 Related Work

In this section, we give an overview on algorithmic results in boundary detection. We focus on work that detects the boundary of the network itself. Intuitively speaking, this refers to the line between densely populated and empty areas of the plane. A different problem with the same name is detecting the boundary of some phenomenon that is monitored by the network. In this problem, two adjacent nodes can identify the boundary by observing that one of them is in the phenomenon's area (say, because some sensor value is above a pre-defined threshold), while the other is not. See [NM03] for such an approach and further pointers. A great survey on classifications for boundaries and holes as they appear in different applications can be found in [AKJ05].

Using Positions: When the nodes' coordinates are available, defining and detecting the boundary is considerably easier. There are several proposals in this settings.

Martincic and Schwiebert [MS04] define v as an inner node, if it is enclosed by a cycle in $N_2(v)$. There are some additional constraints on the cycle's structure. For example, of any two consecutive nodes in the cycle, at least one must be a direct neighbor of v. The boundary consists of all nodes that fail to become inner nodes.

Fang, Gao, and Guibas [FGG04] propose a boundary detection algorithm as a byproduct of their work on geometric routing. They detect nodes where a greedy geometric forwarding scheme would get stuck: v is denoted *strong stuck*, if there exists an $x \in \mathbb{R}^2$ outside of v's communication range, such that no neighbor of v is closer to x than v itself. Furthermore, the authors develop distributed algorithms that construct cycles around the holes at stuck nodes.

Zhang, Zhang, and Fang [ZZF06] consider the coverage area of a network, i.e., the set $A \subset \mathbb{R}^2$ of points that are within the sensing range of at least one node. The boundary is defined as the set of those nodes that are within a certain distance to ∂A. They propose a distributed algorithm utilizing a localized Voronoi diagram, and another algorithm similar to [MS04] above. Both require only knowledge about $N_1(v)$ to decide whether v is a boundary node.

Ahmed, Kanhere, and Jha [AKJ06] propose a distributed algorithm that detects

the outer boundary. Starting at an extreme node, the perimeter is explored with a local right-hand rule. The authors do not consider how to identify inner holes.

Without Positions: In the context of this thesis, distributed algorithms that do not rely on existing location knowledge are far more important.

Funke [Fun05] proposes the following idea: Consider an arbitrary circle $L := \{x \in \mathbb{R}^2 : \mathrm{d}(x, x_0) = h\}$ around x_0 and intersect it with the network area A. This produces circle segments whose end points are on ∂A. In network terms, this translates to the following algorithm: Fix a number of vertices and compute the h-neighborhoods for some h. Within each neighborhood component, find the extreme nodes and mark them as boundary. This was further extended to an improved algorithm by Funke and Klein [FK06b]: The vertices becoming circle centers are a maximal independent set. For each, inspect the set $L \subset V$ of nodes at hop distance h. If this set is connected, attempt to find two nodes at maximum distance. If these two can be separated by removing a 2-hop neighborhood from L, conclude that L is no full circle, hence they are at the ends of a circle segment. These become the boundary nodes. The algorithm can easily be distributed, because it does not use any global information. The computation for a center node v uses only $N_h(v)$.

Wang, Gao, and Mitchell [WGM06] propose another algorithm rooted in the continuous case. Starting from an arbitrary node, they build a tree serving as a discrete version of the shortest path map [Mit91]. Then, a cycle enclosing many holes is constructed. It is split into many cycles surrounding individual holes. Nodes at maximal (as opposed to maximum) distance to these cycles are the initial boundary nodes, which are then connected to form cycles along the boundary. The authors prove correctness for the continuous case; the behavior in discrete networks is left to simulations. They claim the algorithm to work in many networks as long as there is a correlation between communication links and distances. However, it depends on the witness property (Lemma 2.4), which ties the algorithm to $\sqrt{2}/2$-QUDGs.

3.2 Exploiting Uniformity

This section describes our first approach to boundary detection. It was published in [FKP+04]. An improved variant in the same spirit can be found in [FKKL05]. The idea is the following: If the nodes are distributed uniformly, a node at the boundary will have less neighbors in expectation, because part of its communication range is in the void. We assume that the positioning of nodes in the region is the result of a random process with a uniform distribution over A. Furthermore, we assume the communication graph to be a UDG.

Using the notation $V(A) := \{v \in V : v \in A\}$, the expected number of nodes to fall

into an area $A' \subseteq A$ is therefore

$$\mathrm{E}\left[\left|V(A')\right|\right] = n\frac{\lambda(A')}{\lambda(A)}\,, \tag{3.1}$$

where λ denotes the Lebesgue measure in $\mathbb{R}^2$. Therefore, a node $v \in V$, whose distance to ∂A exceeds 1, i.e., $B_1(v) \subset A$, has an estimated neighborhood size of

$$\mu := \mathrm{E}[|N(v)|] = (n-1)\frac{\pi}{\lambda(A)}\,. \tag{3.2}$$

Recall that Chebychev's inequality shows that for a binomial distribution for n events with probability p, i.e., for a $\mathrm{bin}(n,p)$-distributed random variable X, and $\alpha < 1$,

$$\Pr\left[X \leqslant \alpha np\right] \leqslant \frac{1}{n} \cdot \mathrm{const} \to 0 \; (n \to \infty) \tag{3.3}$$

holds. We exploit this fact to provide a simple local rule to let nodes decide whether they are close to the boundary ∂A. Let $\alpha < 1$ be a fixed parameter, and let

$$D = D(\alpha) := \{v \in V : |N(v)| \leqslant \alpha\mu\}\,. \tag{3.4}$$

To show that D actually reflects what we consider the boundary, we present two theorems. The first one states that nodes that are far from ∂A are not in D:

Theorem 3.1. *Let v be a node whose communication range lies entirely in A. Then $v \notin D$ with high probability.*

Proof. This follows directly from (3.3), as

$$\Pr\left[|N(v)| \leqslant \alpha\mu\right] = \Pr\left[\left|V(B_1(v)) \setminus \{v\}\right| \leqslant \alpha\mu\right] \to 0 \quad (n \to \infty)\,. \tag{3.5}$$

□

The second important property of D is that every point on ∂A has a nearby node that belongs to D:

Theorem 3.2. *Let $x \in \partial A$ be on the network area's boundary. Let $\varepsilon > 0$. Assume $\alpha > \frac{1}{\pi}\lambda(B_{1+\varepsilon}(x) \cap A)$. Then, with high probability, there is a node $v \in D$ with $\mathrm{d}(x,v) \leqslant \varepsilon$.*

Proof. Let $A_\varepsilon(x) := B_\varepsilon(x) \cap A$ be the area where v is supposed to be. Then $\lambda(A_\varepsilon(x)) > 0$ by our assumption on feature size. The probability that there is no node in $A_\varepsilon(x)$ equals the probability for a $\mathrm{bin}\left(n, \frac{\lambda(A_\varepsilon(x))}{\lambda(A)}\right)$-distributed variable to become zero, i.e.,

$$\Pr\left[\left|V(A_\varepsilon(x))\right| = 0\right] = \left(1 - \tfrac{\lambda(A_\varepsilon(x))}{\lambda(A)}\right)^n \to 0 \quad (n \to \infty)\,. \tag{3.6}$$

Algorithm 3.1: Heuristic for stochastic boundary detection

1 Obtain Δ via ConvergeCast from v_0
2 Estimate μ via ConvergeCast from v_0
3 Broadcast $\{\alpha_1, \ldots, \alpha_k\}$, report boundary counts to v_0
4 v_0 chooses and broadcasts final α
5 Every node v with $|N(v)| \leqslant \alpha\mu$ marks itself as "boundary node"

On the other hand, the probability that a node u in $A_\varepsilon(x)$ has more than $\alpha\mu$ neighbors is

$$\begin{aligned} \Pr\big[|N(u)| > \alpha\mu\big] &= \Pr\Big[\big|V(A_\varepsilon(x))\big| > \alpha\mu + 1 \mid u \text{ exists}\Big] && (3.7)\\ &\leqslant \Pr\Big[\big|V(B_{1+\varepsilon}(x))\big| > \alpha\mu + 1\Big] && (3.8)\\ &\to 0 \quad (n \to \infty), \text{ because } \alpha\lambda_\circ > \lambda(A_{R+\varepsilon}(x))\,. && (3.9) \end{aligned}$$

Together, we get

$$\begin{aligned} & \Pr\big[\exists v \in V : v \in A_\varepsilon(x), |N(v)| \leqslant \alpha\mu\big] && (3.10)\\ = \;& 1 - \Pr\big[V(A_\varepsilon(x)) = \varnothing\big] \\ & - \Pr\big[\forall v \in V(A_\varepsilon(x)) : |N(v)| > \alpha\mu \mid V(A_\varepsilon(x)) \neq \varnothing\big] && (3.11)\\ \geqslant \;& 1 - \Pr\big[V(A_\varepsilon(x)) = \varnothing\big] - \Pr\big[|N(v)| > \alpha\mu \mid v \in V(A_\varepsilon(x))\big] && (3.12)\\ \to \;& 1 \quad (n \to \infty)\,, && (3.13) \end{aligned}$$

which proves the claim. □

The assumed lower bound on α can be derived from natural geometric properties. For example, if all angles are between $\frac{\pi}{2}$ and $\frac{3\pi}{2}$, then for $\alpha > 0.75$ the condition holds for a reasonably small ε. We conclude that D reflects the boundary very closely. It can be determined by a simple local rule, namely checking whether the number of neighbors falls below $\alpha\mu$. However, this requires that all nodes actually know the value of $\alpha\mu$.

3.2.1 Distributed Heuristic

We use a simple heuristical approach to find the boundary. See Algorithm 3.1 for an overview. The next sections focus on these issues by providing distributed methods for estimating μ and α. We assume there is a node $v_0 \in V$ to steer the different phases of the algorithm.

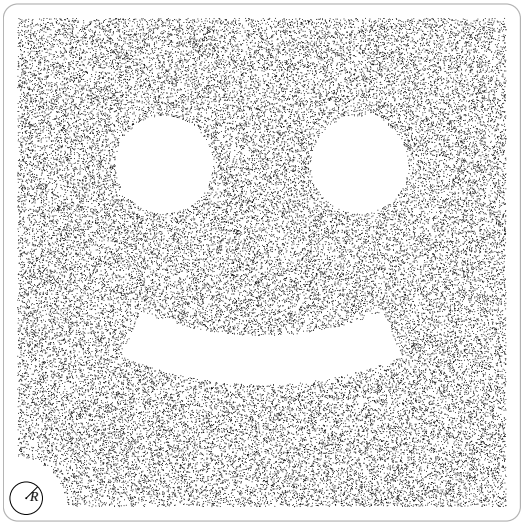


Figure 3.1: Example network. *This network features 45 000 nodes, with a uniform distribution and* $\mu \approx 180$.

Estimate μ (Steps 1 and 2): The density μ is the average degree over *unconstrained* neighborhoods, i.e., nodes that are not close to the boundary. Unfortunately, the nodes do not know whether they are boundary nodes in the beginning of our boundary detection algorithm. Instead, the root collects a constant-size degree histogram via a ConvergeCast. First, it queries the network for Δ, the maximum degree. It then slices the range $[0, \Delta]$ into c_1 slots, where c_1 is a constant. Now, it counts the number of nodes with matching degree for every slot; this is done using a second ConvergeCast. The degree histogram arises by overlaying three different distributions:

1. The neighborhood sizes of all non-boundary nodes.

2. The neighborhood sizes of near-boundary nodes, at varying distance from the boundary.

3. The neighborhood sizes of boundary nodes.

So, for networks where most nodes are non-boundary, we expect a pronounced binomial distribution around μ for (1), a uniform distribution for values safely between $\mu/2$ and μ for (2), overlayed with a small binomial distribution for values under $\mu/2$ for (3), possibly skewed in the presence of many nodes near corners of the region. Consider the network in Figure 3.1. It consists of 45 000 nodes. Figure 3.2 shows the degree histogram for this network. Obviously, it resembles the expected shape very closely.

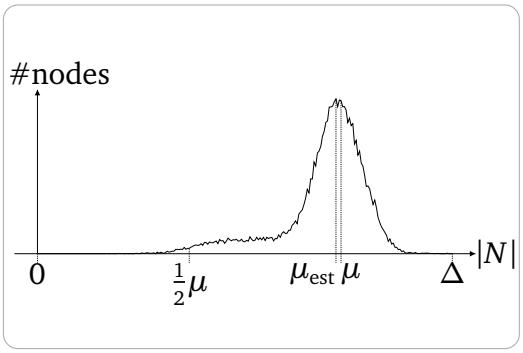


Figure 3.2: Node degree histogram. *The peak μ_{est} is a good approximation for the unconstrained average degree μ.*

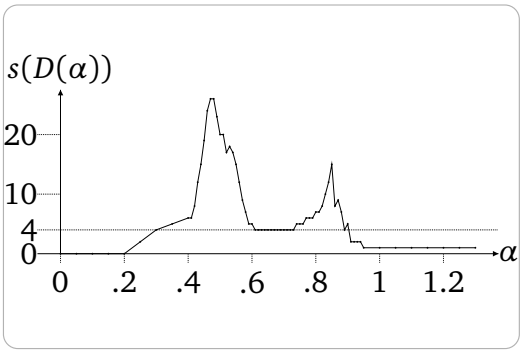


Figure 3.3: Number of boundary components as a function of α. *For $\alpha \approx .7$, the correct number of boundaries is found.*

Once the root receives the histogram, it determines the peak of it and selects a degree μ_{est} from that slice as an estimate for μ. In our example, $\mu \approx 179.7$ and $\mu_{\text{est}} = 177$.

Estimate α (Steps 3 and 4): Our algorithms depend on a good choice of the area-dependent parameter α, which should be as low as possible without violating the lower bound from Theorem 3.2. If a bound on corner angles is known in advance, say, $3\pi/2$ in a rectilinear setting, this is easy: For example, choose α slightly larger than $3/4$. As this may not always be the case, it is desirable to develop methods for the swarm itself to determine a useful α.

Assume the correct value of α was known. Then, the root could simply broadcast this value together with μ_{est}, and every node v with $|N(v)| \leqslant \alpha\mu_{\text{est}}$ would mark itself as a boundary node. Because α is not known, the root chooses a set of c_2 possible values $\alpha_1, \ldots, \alpha_{c_2} \in [0,1]$ and broadcasts them. Every node v then joins $D(\alpha_i)$ for all $i \in \{1, \ldots, c_2\}$ with $|N(v)| \leqslant \alpha_i\mu_{\text{est}}$.

Now, the network needs to find out which of the α_i produces the "best" boundary. For that matter, the network counts the number $s(D(\alpha_i))$ of distance-2 connected components in $D(\alpha_i)$, for all i. This is done by running c_2 spanning tree algorithms in parallel, one in each $D(\alpha_i)$. These algorithms run on N_2, in the sense that two nodes u and v are considered adjacent if $v \in N_2(u)$, for added stability in thin parts of the boundary. The ConvergeCast then continues and reports the number of roots produced by each algorithm, equalling $s(D(\alpha_i))$ for every i, back to v_0.

Now v_0 chooses one α_i for the final boundary construction, based on the following observation: For a too small α, $D(\alpha) = \emptyset$. For increasing α, the number of connected boundary pieces grows rapidly, until α is large enough to allow different pieces of the same boundary to grow together, eventually forming the correct set of boundary strips. When further increasing α, additional boundaries appear in

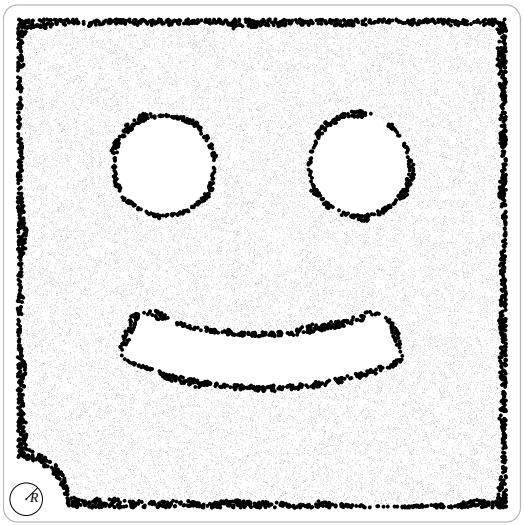


Figure 3.4: Experimental results for the example network. *With such a high density, the marked nodes describe the boundary accurately.*

low-density areas, increasing the number of identified boundaries. These boundaries also begin to merge, until eventually a single boundary consisting of the whole network is left. Figure 3.3 shows that this expected behavior does indeed occur in reality: Notice the clear plateau at 4 connected components, embedded between two pronounced peaks. So v_0 identifies an $i^* \in \{0, \ldots, c_2\}$ such that $s(D(\alpha_{i^*}))$ is a local minimum between two maxima. This value is then broadcast to all nodes.

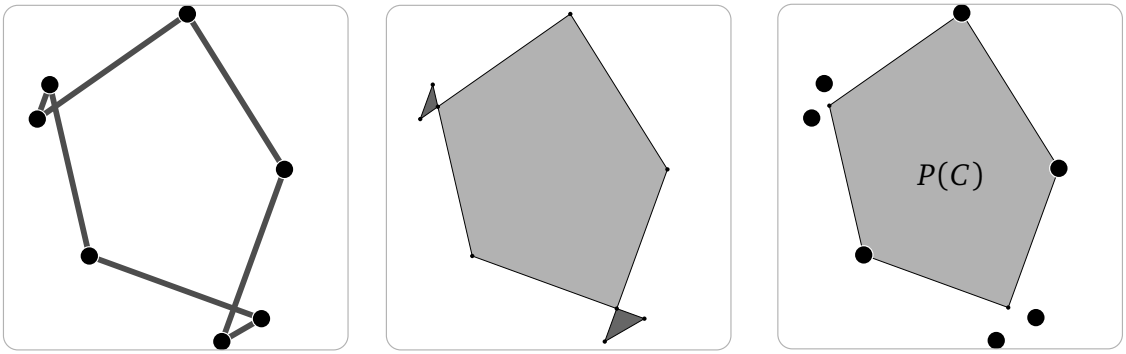
Establish final boundary (Step 5): As all nodes already constructed the boundary for the final choice of i^*, they merely have to drop the now obsolete information about the other α_i, and the boundary detection is complete.

Running this algorithm on the example network results in the boundary shown in Figure 3.4.

While the algorithm produces nice results on the shown network, it does require a ridiculously high density. In the simulation, we used $\mu \approx 179.7$. It does however demonstrate that detecting the boundary by exploiting the distribution is possible.

3.3 Deterministic Boundaries

In the following sections, we propose another approach to detect the network's boundary without using coordinates. It was originally published as [KFPF06], and there is a demonstrational video [FK06a] about it. What separates it from the algorithm in the previous section, as well as the work summarized in Section 3.1.1,



(a) Cycle C in d-QUDG. (b) Faces of straight-line embedding. (c) Polygon $P(C)$ without artefacts.

Figure 3.5: Extracting $P(C)$ from C. *While C itself does not induce a simple polygon, removing triangular artefacts leads to the desired $P(C)$.*

are the provable properties of the solutions: The algorithm produces polygons, described implicitly by cycles in the network, and identifies a set of *inner* nodes such that, whatever the actual embedding p is, all of these nodes must be located on the inside of the polygons. In other words, the cycle nodes are on the boundary of a network area populated by the inner nodes.

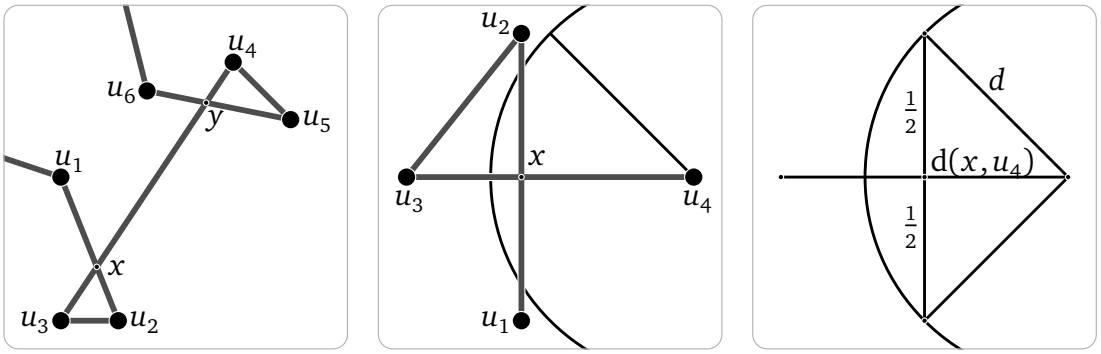
Our algorithm's aim is to find such a structure that spans the whole network. But even in cases where it does not succeed, it's output is still a valuable description of network topology, because it not only contains some nodes that are presumably "boundary", but also the sub-network to which these nodes are indeed the boundary. Hence, the algorithm is well-suited to node distributions where it is hard to actually define the boundary—it will automatically identify regions where it can prove something.

First we give a rigorous definition of the structures we are looking for. Then we provide criteria that prove topological properties, independent of the actual embedding p. We will then develop distributed algorithms to find boundary structures, and evaluate them using simulations.

3.3.1 Definitions

Our aim is to use cycles in the network to describe polygons, and to prove that some nodes are located within it. Let $C = (v_1, \ldots, v_k, v_1)$ induce a chordless cycle in G. Consider the straight-line embedding of the cycle, according to the embedding p. While this is not necessarily a simple polygon, we can extract one from it. It is denoted by $P(C)$. See Figure 3.5.

From the witness property (Lemma 2.4) and the fact that C is chordless we can



(a) (b) (c)

Figure 3.6: Artefact triangles are faces. *Three visualizations used to prove that artefacts are faces in the straight-line embedding of C.*

conclude that crossings in the embedding only happen between the edges adjacent to a common cycle edge, i.e., a crossing of cycle edges uv and wx is impossible unless there is an edge connecting $\{u, v\}$ with $\{w, x\}$.

Lemma 3.3. *Let $u_0, \ldots, u_6$ be consecutive nodes in C. Assume the lines u_1u_2 and u_3u_4 cross in a point x. Then the* artefact triangle *xu_2u_3 is a face of C's embedding.*

Proof. See Figure 3.6. Assume xbc is no face of the embedding. This is only possible if another line crosses the triangle. Because C is a chordless cycle, this line must belong to a cycle edge. The Witness Property guarantees that any crossing edges must have a common adjacent edge, i.e., crossing edges have distance 2 in the cycle.

The edge u_2u_3 can only be crossed by u_0u_1 and u_4u_5. Due to the triangle's orientation, this is impossible because they would need to cross u_1u_2 resp. u_3u_4 as well, see Figure 3.6(a). The line u_3u_4 can only cross u_1u_2 (which it does) and u_5u_6. So assume it crosses both, and let y be the intersection point with u_5u_6. We claim that x is closer to u_3 than y is, thereby proving that u_5u_6 does not intersect the triangle. The distance $\mathrm{d}(x, u_4)$ is the distance between a point on a line between two points (u_1 and u_2) that have distance from u_4 at least d, see Figure 3.6(b). A simple application of Pythagoras' Theorem (see Figure 3.6(c)) shows that any such point has a distance of at least $1/2$ to u_4, hence $\mathrm{d}(x, u_4) \geqslant 1/2$. Due to $\mathrm{d}(u_3, u_4) \leqslant 1$, we get $\mathrm{d}(x, u_3) \leqslant 1/2$.

Using the same argument, it can be shown that $\mathrm{d}(y, u_3) \geqslant 1/2$. As $|C| > 6$, u_1u_2 and u_5u_6 cannot cross, and hence $x \neq y$. Hence $\mathrm{d}(x, u_3) < \mathrm{d}(y, u_3)$, proving that no line enters the triangle over the side u_3x. The same argument can be repeated for u_2x. □

Before we use the above lemma to show how to use cycles to describe areas in

the plane, let us present a trivial corollary that we will need to separate nodes in these areas from the outside.

Corollary 3.4. *In the setting of Lemma 3.3, assume a node is located within the triangle artefact xu_2u_3. Then this node is adjacent to u_2 or u_3.*

Proof. This corollary follows directly from the proof of Lemma 3.3, where we show that $\mathrm{d}(x,u_2) \leqslant 1/2$ and $\mathrm{d}(x,u_3) \leqslant 1/2$. Therefore, all points in the triangle have distance at most $1/2$ to at least one of the two nodes. Because of the standing assumption $d \geqslant \sqrt{2}/2 > 1/2$, the claim follows. □

We have seen that a cycle's straight-line embedding does not necessarily define a simple polygon. Fortunately, Lemma 3.3 shows how to obtain one, see Figure 3.5(c):

Definition 3.5. *Let C induce a chordless cycle in G and consider its straight-line embedding. The polygon obtained by removing all triangles from Lemma 3.3 is denoted by $P(C)$.*

To describe a region in the plane, we use multiple chordless cycles in the graph that follow the perimeter of the region. There is always one cycle for the outer perimeter. If the region has holes, there is an additional cycle for each of them.

Definition 3.6. *Let $A \subset \mathbb{R}^2$ be a compact region. Let $\mathscr{C} \subset 2^V$ be a finite family, where*

(i) every $C \in \mathscr{C}$ induces a chordless cycle, and

(ii) no two nodes from different cycles are adjacent.

We say $\mathscr{C}$ describes *A and write $A = A(\mathscr{C})$, if $\partial A = \bigcup_{C\in\mathscr{C}} \partial P(C)$. $\mathscr{C}$ is called* boundary cycle family*, if $A(\mathscr{C})$ exists.*

A word is necessary why $A(\mathscr{C})$ is well-defined. Consider a given set of cycles. Condition (ii) ensures that the polygon $P(C)$ for different $C \in \mathscr{C}$ do not intersect. Now let A be the closure of all points that are inside an odd number of these areas. It is easy to see that $A = A(\mathscr{C})$, using the facts that replacing "odd" with "even" violates the requirement that A is bounded, and that there is no freedom in choosing individual points from $\bigcup_{C\in\mathscr{C}} \partial P(C)$ because A must be closed.

Our goal is to find the boundary of the network, which supposedly separates the "inside" of the network from the "outside" and "holes". It is thus necessary to define what a hole actually is:

Definition 3.7. *Consider the straight-line embedding of G. It defines a decomposition of the plane into faces. A finite face F of this decomposition is called* h-hole *for parameter h, if the boundary length of the convex hull of F strictly exceeds h.*

There are many possibilities to define a hole, and most of them involve a threshold parameter. The important property of an h-hole F, using our definition, is the following trivial fact: Let C induce a chordless cycle with $|C| \leqslant h$. Then all points $f \in F$ are on the outside of $P(C)$. This provides a simple way to prove that a given cycle does not embrace holes. Our boundary algorithm hinges on this property: The final boundary will consist of cycle pieces that stem from a covering of the area with small cycles, proving that there is no large hole in the area that is marked "inside" by our algorithm.

Using the previously defined concepts, we now give a concise definition of the structures we are looking for:

Definition 3.8. *A* feasible geometry description *(FGD) is a pair $(\mathscr{C}, I)$, where*

(i) $\mathscr{C} \subset 2^V$ *is a boundary cycle family (Definition 3.6),*

(ii) $I \subset V$ *is a set of nodes with* $p(v) \in A(\mathscr{C})$ *for all* $v \in I$*, and*

(iii) $A(\mathscr{C})$ *does not contain an inner point of any h-hole for* $h > K$*, where* K *is a given constant.*

The motivation behind this definition is the following: The boundary cycle family $\mathscr{C}$ defines an area $A(\mathscr{C})$ that is densely populated by nodes, in the sense that it does not contain holes. The set I contains only "inside" nodes, hence $|I|$ is an indicator for the size of $A(\mathscr{C})$. See Figure 3.15 (page 60) in Section 3.4 for different FGDs in an example network.

Note that this definition allows for a very memory-efficient type of topology knowledge: Every node needs only two bits, encoding its membership in I resp. some boundary cycle $C \in \mathscr{C}$, without the need to store which C it is in, because the cycles are mutually disconnected.

The constant K defines how large a hole must be to be detected; it serves as a threshold to separate areas that are thin-populated from topologically relevant holes. When looking at straight-line drawings, people generally agree that 20-holes are real holes, and 10-holes look like fluctuations in density. Hence, we fix $K = 15$ for our experiments; the algorithm works for any $K > 6$. When $K \leqslant 6$, the intermediate structures we use in the algorithm no longer exist.

Because $|I|$ is an indicator for the area that is spanned by an FGD, we are looking for one with maximum I. This forces the boundary cycles to follow the actual network boundary as close as possible. The optimization problem we consider for boundary recognition is therefore

$$\text{(BD)} \begin{cases} \max & |I| \\ \text{s.t.} & (\mathscr{C}, I) \text{ is an FGD.} \end{cases} \tag{3.14}$$

Additionally, we want to solve this problem without using the embedding p, hence without the ability to determine $A(\mathscr{C})$. Instead, the algorithm restricts itself to solutions that are FGDs in *every* network embedding as a d-QUDG simultaneously.

This is done using structures that have topological properties that are invariant to p. These will be introduced in the following sections.

3.3.2 Fit Numbers

Our goal is to prove that certain nodes are located *inside* some region $P(C)$. As a first step, we show how to prove that they are *outside* $P(C)$. The Witness Property allows to use edges in the graph to separate nodes in the embedding p, even without knowing p.

Lemma 3.9. *Let C be a chordless cycle in G, and let $U \subseteq V \setminus (C \cup N(C))$ be a connected node set. Then either all nodes in U are in $P(C)$, or none of them is.*

Proof. Assume there is a node in $P(C)$, and one outside of it. Then there is also a pair $i, o \in U$ that is adjacent, with $i \in P(C)$ and $o \notin P(C)$. Node o is not in an artefact triangle due to Corollary 3.4. The line between i and o leaves the triangle, hence crosses a triangle edge, all of which correspond to cycle edges. Hence i or o must be in $N(C)$, which is a contradiction. □

Note that simply using two node sets that are separated by a chordless cycle C and proving that the first set is outside the cycle does not guarantee that the second set is on the inside. The two sets could be on different sides of $P(C)$. So we need more complex arguments to certify insideness.

First, we present a certificate for being on the outside. Define $\mathrm{fit}_d(n)$ to be the maximum number of independent nodes $j \in J$ that can be placed inside a chordless cycle C of at most n nodes in any d-QUDG embedding such that $J \cap N(C) = \emptyset$. We say that nodes are independent, if there is no edge between any two of them. These numbers exist because independent nodes are always placed at a distance of at least d from each other, so there is a certain area needed to contain the nodes. On the other hand, C defines a polygon of perimeter at most $|C|$, which cannot enclose arbitrarily large areas. Also we define $\mathrm{enc}_d(m) := \min\{n : \mathrm{fit}_d(n) \geqslant m\}$, the minimum length needed to fit m nodes.

Definition 3.10. *A pair (K, N) of integers is called d-*realizable*, if there exists a simple polygon $P \subset \mathbb{R}^2$ with vertices $p_1, \ldots, p_N$, and points $q_1, \ldots, q_K \in P$ such that the following condition holds:*

Consider the set of $N + K$ circles $\{B_{d/2}(p_1), \ldots, B_{d/2}(p_N), B_{d/2}(q_1), \ldots, B_{d/2}(q_K)\}$. Then any two of them are either disjoint, or they belong to consecutive vertices of the polygon.

See Figure 3.7 for an example that shows that $(2, 10)$ is 1-realizable. This allows for a straightforward definition of the fit number:

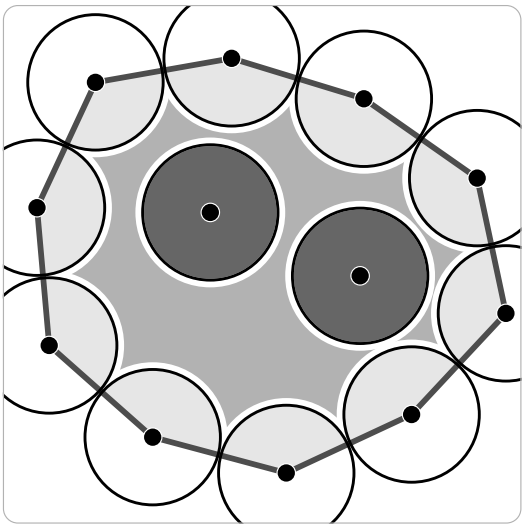


Figure 3.7: Feasible 1-realization for $(2, 10)$. *Note that the circles at the polygon's vertices are touching.*

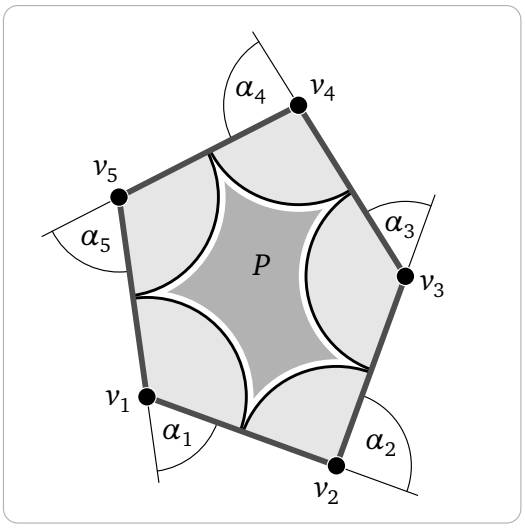


Figure 3.8: Angles in a realization. *The size of the inner segments is defined by the direction change of the walk.*

Definition 3.11. $\mathrm{fit}_d(N) := \max\{K : (K, N)$ *is d-realizable*$\}$.

It is obvious that $\mathrm{fit}_d(N) = \Theta(N^2)$. However, our boundary detection algorithm (Section 3.3) requires close upper bounds for the actual values, as provided by the following lemma:

Lemma 3.12. $\mathrm{fit}_d(N) \leqslant \frac{1}{2\sqrt{3}\pi d^2} N^2 - \frac{1}{2} N + 1$.

Proof. Let $K = \mathrm{fit}_d(N)$. Let $P \subset \mathbb{R}^2$, $p_1, \ldots, p_N \in \mathbb{R}^2$ and $q_1, \ldots, q_K \in \mathbb{R}^2$ be a realization as described in Definition 3.10 above. First note that we can assume P to be convex, and with all edges having unit length. This follows from the following two facts:

1. When P is not convex, one can find a concave piece of the perimeter and flip it to the outside. This does not violate Definition 3.10, but strictly increases the area of P.

2. When P is convex, but some edge is shorter than 1, one vertex of this edge can be shifted to the outside, again increasing the area without violating Definition 3.10.

Repeating these two operations converges because the area of P is bounded from above. The perimeter of P has length N. Hence the area $\lambda(P)$ of P is at most that of a circle with circumsphere N:

$$\lambda(P) \leqslant \pi(\tfrac{N}{2\pi})^2 = \tfrac{1}{4\pi} N^2 \,. \tag{3.15}$$

Now we identify two disjoint sets that are contained in P. See Figure 3.7. The first is $P_1 := B^\circ_{d/2}(q_1) \dot\cup \ldots \dot\cup B^\circ_{d/2}(q_K)$ (the dark circles in Figure 3.7). Because of the points' minimum distance, these circles are pairwise disjoint, and hence $\lambda(P_1) = K\frac{\pi}{4}d^2$.

The second set is $P_2 := (B^\circ_{d/2}(p_1) \dot\cup \ldots \dot\cup B^\circ_{d/2}(p_N)) \cap P$ (the light circle segments in Figure 3.7). From the minimum distance of points, and that $d/2 \leqslant 1/2$ where all polygon edges have length 1, it follows that the circles are disjoint and $P_1 \cap P_2 = \varnothing$.

Now consider a walk around the polygon, starting at p_1, and finally returning to p_1 over p_N. Assume the walk is counter-clockwise, the other case is analogous. At vertex p_i, the walk makes a turn to the left in relative direction α_i for some $0 \leqslant \alpha_i < \pi$. See Figure 3.8 for a visualization. So $B^\circ_{d/2}(p_i) \cap P$ is a $(\pi - \alpha_i)$-segment of $B^\circ_{d/2}(p_i)$ with area

$$\lambda(B^\circ_{d/2}(p_i) \cap P) = \frac{\pi - \alpha_i}{2\pi}\frac{d^2}{4}\pi. \tag{3.16}$$

The complete walk turns around once and finishes in the same direction as its started, hence $\sum_{i=1}^{N} \alpha_i = 2\pi$. This gives

$$\lambda(P_2) = \sum_{i=1}^{N} \lambda(B^\circ_{d/2}(v_i) \cap P) = \frac{d^2}{8}\sum_{i=1}^{N}(\pi - \alpha_i) = (N-2)\frac{\pi}{8}d^2\,. \tag{3.17}$$

Because $P_1 \dot\cup P_2 \subseteq P$, $\lambda(P_1) + \lambda(P_2) \leqslant \lambda(P)$ must hold. Solving this for K gives

$$K \leqslant \frac{1}{\pi^2 d^2}N^2 - \frac{1}{2}N + 1\,. \tag{3.18}$$

□

To get some insight on how close this bound actually is, we prove a lower bound for $\mathrm{fit}_d(N)$ by presenting six families of feasible realizations. The solutions are for the case $d = 1$, but apply for all d because $\mathrm{fit}_d(N)$ is monotonically increasing with decreasing d.

The solutions begin with a small realizable (K_0, N_0) that can be expanded. The following lemma shows how such a sequence is turned into a lower bound for $\mathrm{fit}_d(N)$:

Lemma 3.13. *Let $K_0, N_0 \in \mathbb{N}$. Let $(K_j, N_j)_{j\in\mathbb{N}_0}$ be the sequence where $K_j = K_{j-1} + N_{j-1}$ and $N_j = N_{j-1} + 6\ \forall j \in \mathbb{N}$.*

If (K_j, N_j) is d-realizable for every $j \in \mathbb{N}_0$, then $\mathrm{fit}_d(N) \geqslant \frac{1}{12}N^2 - \frac{1}{2}N + (K_0 + \frac{1}{2}N_0 - \frac{1}{12}N_0^2)$ for every N that can be expressed as $N = N_0 + 6j$ with $j \in \mathbb{N}_0$.

Proof. First observe that $N_j = N_0 + 6j$ and $K_j = K_0 + jN_0 + 3j(j-1)$. If $N = N_0 + 6j$, then $j = \frac{1}{6}(N - N_0)$. Now

$$\mathrm{fit}_d(N) \geqslant K_j = K_{\frac{1}{6}(N-N_0)} = K_0 + \tfrac{1}{6}(N - N_0) + \tfrac{1}{12}(N - N_0)(N - N_0 - 6)\,,$$

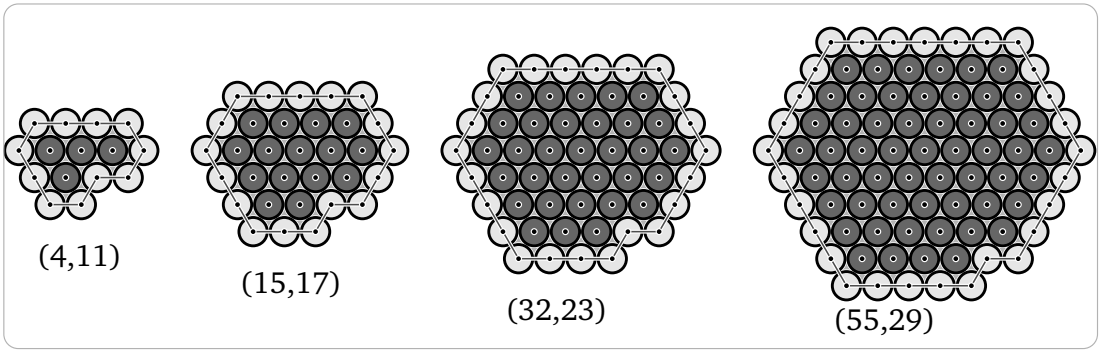


Figure 3.9: Packing sequence for $N_0 = 11$. *Iteratively adding outer layers produces a sequence of packings.*

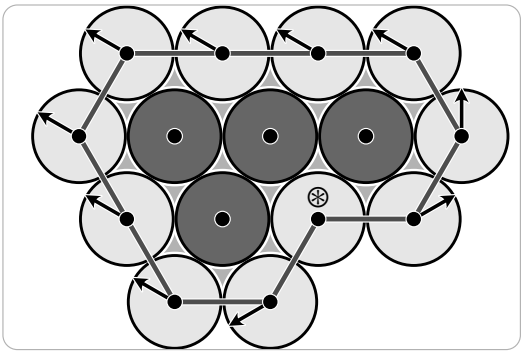


Figure 3.10: Turning the packing into a realization. *The marked node can shifted outwards, thereby allow the other nodes to move slightly in the indicated directions. This makes room for the inner nodes to detach.*

which can be reduced to the claimed bound. □

Consider the hexagonal circle packing shown on the left of Figure 3.9. Note that it is no feasible realization, because the inner circles are not isolated. This packing can be extended: Declare all circles as inner circles, and add a layer of outer circles that consists of all circles in the infinite hexagonal packing that touch an inner circle. This can be iterated, see Figure 3.9. Each outer layer has 6 more circles than the previous one. Hence, if these packings were 1-realizable, $(4, 11)$ would define a valid start for Lemma 3.13. Each of these packings can be transformed into a feasible realization. Figure 3.10 visualizes the transformation for $(4, 11)$. Every packing has a single inwards-placed circle in the lower right corner. This circle can be moved in direction $-60°$, which decreases the length of the edges to the neighboring polygon vertices below 1. Now all other outer nodes can be moved a small amount in the direction of the tangent to some touching inner circle. The outer circles will still be connected, but detached from the inner ones. Now the centers of the inner circles can be scaled by a small amount, detaching them from

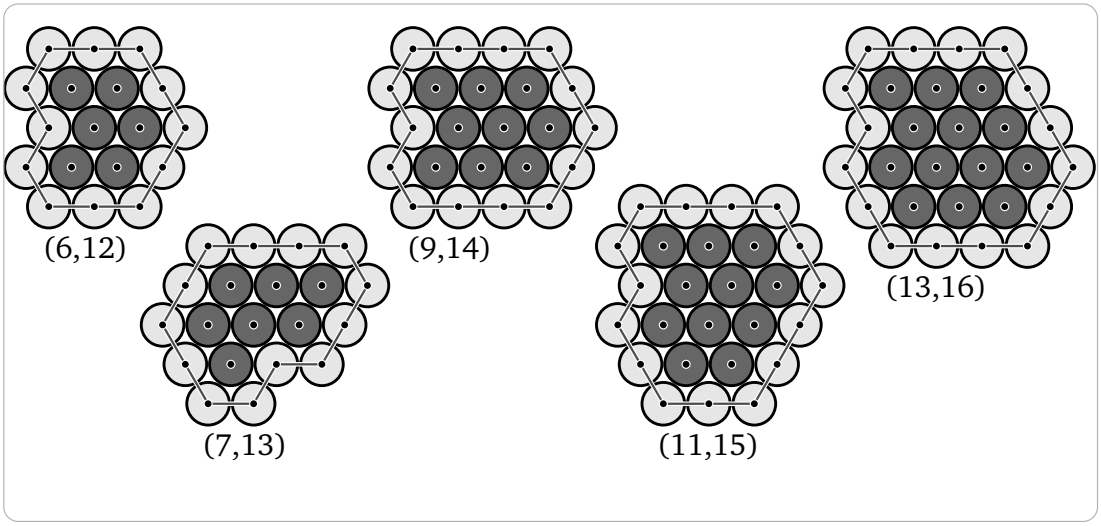


Figure 3.11: Packing sequences. *Each shown packing can be turned into the start of a sequence for Lemma 3.13.*

each other. The result is a feasible realization. This sequence of realizations starts with $(4, 11)$, so this proves the following lemma:

Lemma 3.14. *For $N = 11 + 6j$, $j \in \mathbb{N}_0$, $\mathrm{fit}_d(N) \geqslant \frac{1}{12}N^2 - \frac{1}{2}N - \frac{7}{12}$.*

For the other values of N, we use five other packings. They are shown in Figure 3.11. Each of them can be expanded to a family in the same manner as the previous packing, and each has an inset circle that can be used to turn the hexagonal packing into a feasible realization.

These provide feasible starts for Lemma 3.13, which are $(6, 12)$, $(7, 13)$, $(9, 14)$, $(11, 15)$, and $(13, 16)$. Putting everything together, we get the following:

Lemma 3.15. *For $N \geqslant 11$, $\mathrm{fit}_d(N) \geqslant \frac{1}{12}N^2 - \frac{1}{2}N - \frac{7}{12}$.*

Proof. Every $N \geqslant 11$ can be expressed as $N = F + 6j$ for $F \in \{11, 12, \ldots, 16\}$ and $j \in \mathbb{N}_0$. According to Lemma 3.13, $\mathrm{fit}_d(N) \geqslant \frac{1}{12}N^2 - \frac{1}{2}N + c_F$, where c_F is a constant depending on F. The possible values for c_F are $\{-\frac{7}{12}, 0, -\frac{7}{12}, -\frac{1}{3}, -\frac{1}{4}, -\frac{1}{3}\}$. The minimum of these is $-\frac{7}{12}$, hence it is valid for all $N \geqslant 11$. □

To summarize the obtained bounds on fit numbers, we state the following theorem:

Theorem 3.16. *For $N \geqslant 11$,*

$$\left\lceil \tfrac{1}{12}N^2 - \tfrac{1}{2}N - \tfrac{7}{12} \right\rceil \leqslant \mathrm{fit}_d(N) \leqslant \left\lfloor \tfrac{1}{\pi^2 d^2}N^2 - \tfrac{1}{2}N + 1 \right\rfloor .$$

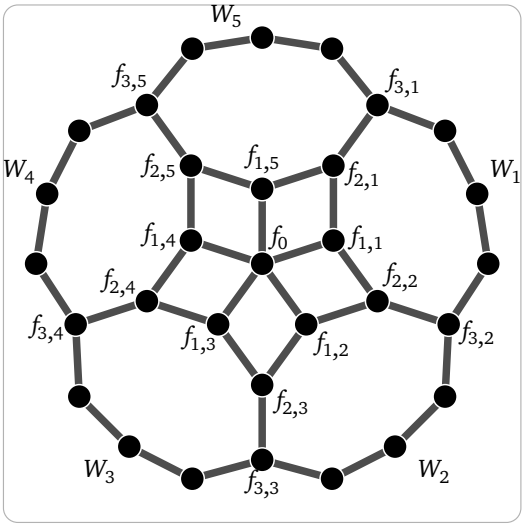


Figure 3.12: A 5-flower. *In every embedding as a $\sqrt{2}/2$-QUDG, the central node is on the inside of the outer cycle.*

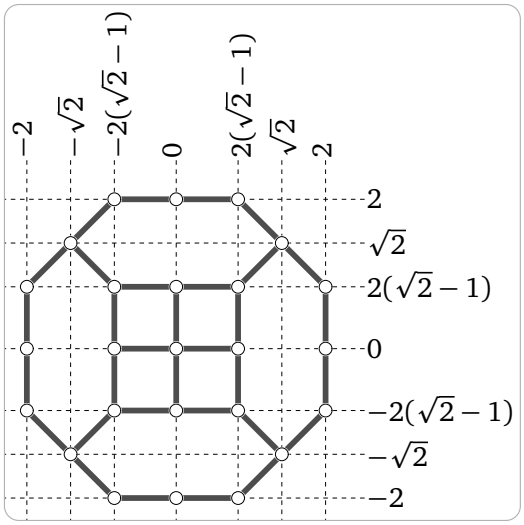


Figure 3.13: Constructing a 4-flower in a dense region. *Picking arbitrary nodes from the shown areas always produces a flower.*

Now we can give a simple criterion to decide that a node set is outside a chordless cycle:

Lemma 3.17. *Let C be a chordless cycle and $I \subset V \setminus N(C)$ be a connected set that contains an independent subset $J \subset I$. If $|J| > \mathrm{fit}_d(|C|)$, then every node in I is outside $P(C)$.*

Proof. By Corollary 3.9 and the definition of fit_d. □

3.3.3 Flowers

So far, we have presented criteria by which one can decide whether some nodes are *outside* a chordless cycle, based on a packing argument. Such a criterion will not work for the *inside*, as any set of nodes that fit in the inside can also be accommodated by the unbounded outside. Instead, we now present a stronger structural criterion that is based on a particular subgraph, an m-flower. For such a structure, we can prove that there are some nodes on the inside of a chordless cycle. Our algorithmic methods start by searching for flowers, leading to an FGD. We begin by actually defining a flower, see Figure 3.12 for a visualization.

Definition 3.18. *An* m-flower *in G is an induced subgraph according to the following constraints:*

- *The node set consists of*
  - *(i) a seed $f_0 \in V$,*
  - *(ii) independent nodes $f_{1,1}, \ldots, f_{1,m} \in V$,*
  - *(iii) bridges $f_{2,1}, \ldots, f_{2,m} \in V$,*
  - *(iv) hooks $f_{3,1}, \ldots, f_{3,m} \in V$,*
  - *(v) and chordless paths $W_1, \ldots, W_m$, where each $W_i = (w_{j,1}, \ldots, w_{j,\ell_j}) \subset V$.*

  All of these $1 + 3m + \sum_{j=1}^{m} \ell_j$ nodes have to be different nodes. For convenience, we define $f_{j,0} := f_{j,m}$ and $f_{j,m+1} := f_{j,1}$ for $j = 1, 2, 3$.

- *The edges of the subgraph are the following:*
  - *(i) The seed f_0 is adjacent to all independent nodes: $f_0 f_{1,j} \in E$ for $j = 1, \ldots, m$.*
  - *(ii) Each independent node $f_{1,j}$ is connected to two bridges: $f_{1,j} f_{2,j} \in E$ and $f_{1,j} f_{2,j+1} \in E$.*
  - *(iii) The bridges connect to the hooks: $f_{2,j} f_{3,j} \in E$ for $j = 1, \ldots, m$.*
  - *(iv) Each path W_j connects two hooks, that is, $f_{3,j} w_{j,1}, w_{j,1} w_{j,2}, \ldots, w_{j,\ell_j} f_{3,j+1}$ are edges in E.*

- *Finally, the path lengths ℓ_j, $j = 1, \ldots, m$ obey*

$$\mathrm{fit}_d(7 + \ell_j) \quad < \quad \left\lceil \frac{1}{2} \left(\sum_{k \neq j} \ell_k + m - 4 \right) \right\rceil . \tag{3.19}$$

Note that flowers actually exist: The flower shown in Figure 3.12 is of the desired structure. It has $m = 5$ and $\ell_1 = \ell_2 = \ldots = \ell_5 = 3$. As $\mathrm{fit}_1(7 + 3) \leqslant 6$ by Theorem 3.16, Equation (3.19) holds.

The beauty of flowers lies in the following fact:

Lemma 3.19. *In every d-QUDG embedding of an m-flower, the independent nodes are placed on the inside of $P(C)$, where $C := \{f_{3,1}, \ldots, f_{3,m}\} \cup \bigcup_{j=1}^{m} W_j$.*

Proof. Let $P_j := (f_{1,j}, f_{2,j}, f_{3,j}, W_j, f_{3,j+1}, f_{2,j+1})$ be a petal of the flower. P_j defines a cycle of length $5 + \ell_j$. The other nodes of the flower are connected and contain $m - 2$ independent bridges. According to (3.19), this structure is on the outside of $P(P_j)$.

Therefore, the petals form a ring of connected cycles, with the seed on either the inside or the outside of the structure. Assume the seed is on the outside. Consider the infinite face of the straight-line embedding of the flower. The seed is part of the outer cycle, which consists of $7 + \ell_j$ nodes for some $j \in \{1, \ldots, m\}$. This cycle has to contain the remaining flower nodes, which contradicts (3.19). Therefore, the seed is on the inside, and the claim follows. □

Corollary 3.20. *Let $\mathscr{C}$ be the set consisting of the cycle nodes, i.e., hooks and paths, of some flowers in G. Let I be the union of their seeds and independent nodes. Suppose there is no edge between nodes of different flowers. Then $(\mathscr{C}, I)$ is an FGD.*

Proof. Lemma 3.19 proves that a single flower is an FGD. The feasibility of $(\mathscr{C}, I)$ is then a direct consequence of the flowers being mutually disconnected. □

Because we do not assume a particular distribution of the nodes, we cannot be sure that there is a flower in the network. Intuitively, this is quite clear, as any node may be close to the boundary, so that there are no interior nodes. As the nodes can only make use of the local graph structure, and have no direct way of detecting region boundaries, this means that our criterion may fail in low densities. As we show in the following, we can show the existence of a flower if there is a densely populated region somewhere: We say G is *locally ε-dense* in a region $A \subset \mathbb{R}^2$, if every ε-disk in A contains at least one node, i.e., $\forall z \in \mathbb{R}^2 : B_\varepsilon(z) \subset A \Rightarrow \exists v \in V : \mathrm{d}(v,z) \leqslant \varepsilon$.

Lemma 3.21. *Let $0 < \varepsilon < \frac{3}{2} - \sqrt{2} \approx 0.086$. If G is a UDG and ε-dense on the disk $B_3(z)$ for some $z \in \mathbb{R}^2$, then G contains a 4-flower.*

Proof. See Figure 3.13. Place an ε-ball at all the indicated places and choose a node in each. Then the induced subgraph will contain precisely the drawn edges. Then $m = 4$ and $\ell_1 = \ldots = \ell_4 = 3$, so for $d = 1$, these ℓ-numbers are feasible. □

3.3.4 Augmenting Cycles

Now that we have an initial FGD in the network, defined by some flowers, we seek an improvement method. For that, we employ *augmenting cycles*. Consider an FGD $(\mathscr{C}, I)$. Let $U = (u_1, u_2, \ldots, u_k) \subset V$ be a (not necessarily chordless) cycle. For convenience, define $u_0 := u_k$ and $u_{k+1} := u_1$.

When augmenting, we open the cycles in $\mathscr{C}$ where they follow U, and reconnect the ends according to U. Let $U^- := \{u_i \in U : u_{i-1}, u_i, u_{i+1} \in C\}$ and $U^+ := U \setminus C$. The resulting cycle nodes of the augmentation operation are then $C' := C \cup U^+ \setminus U^-$. If $N(U) \cap I = \varnothing$, this will not affect inside nodes, and it may open some new space for the inside nodes to discover. In addition, as the new cycle cannot contain a $k+1$-hole, we can limit $k < K$ to guarantee condition (iii) of Definition 3.8.

To ensure feasibility of the resulting FGD, we have to make sure that the augmenting cycle does not wrap around a structure, connecting the inner nodes to the outside, see Figure 3.14. This is done by computing a maximal independent set $J \subset I$. If there is a connected component $I' \in I$ with $|J \cap I'| > \mathrm{fit}_d(k)$, the augmenting cycle cannot wrap around this component. By keeping track of which components of I could be affected when augmenting, our algorithm makes sure such a wrap-around never happens.

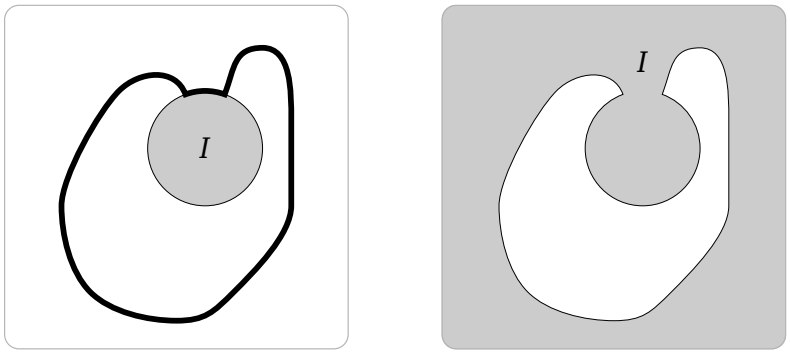


Figure 3.14: Augmenting Cycle wrap-around. *Augmenting with such a cycle would allow I to sweep over almost the whole network, making the FGD meaningless.*

3.3.5 Distributed Algorithm

We maintain a global FGD, decomposed into one or more smaller FGDs $(\mathscr{C}_f, I_f)_{f\in F}$, according to these invariants:

1. Every $(\mathscr{C}_f, I_f)_{f\in F}$ is an FGD.
2. $(\cup_{f\in F}\mathscr{C}_f, \cup_{f\in F} I_f)$ is an FGD.
3. The FGDs are pairwise disconnected, i.e.,

$$\forall f, f' \in F, f \neq f' : u \in (\cup_{C\in\mathscr{C}_f} C \cup I_f), v \in (\cup_{C\in\mathscr{C}_{f'}} C \cup I_{f'}) \Longrightarrow uv \notin E\,.$$

To simplify notation, we will sometimes call an index $f \in F$ an FGD, when we actually mean $(\mathscr{C}_f, I_f)$. In addition to the FGDs, we maintain an independent set $J \subset \cup_{f\in F} I_f$. This set is used to measure the size of an FGD. Initially, the algorithm needs to be careful not to use an augmenting cycle that wraps around an FGD. Later, when the FGD have grown, this is no longer possible due to the length limit on augmenting cycles.

For that matter, we say an FGD $f \in F$ is *small*, if

$$|J \cap I_f| \leqslant \mathrm{fit}_d(K)\,. \tag{3.20}$$

Otherwise, f is *large*. The intuition behind this is that all cycles our algorithm considers have length at most K. Hence, if an FGD is large, the cycle cannot wrap around it.

Nodes store information depending on their role r_v in the FGDs. The roles and associated data are as follows:

"cycle": All nodes $v \in C \in \mathscr{C}_f$ for some $f \in F$ have this role. v stores

- its role $r_v =$ cycle,
- the neighbors that also have this role (always exactly two),
- whether f is small or large, and
- $M(f) := |J \cap I_f|$, if f is small.

"cut": All neighbors of cycle nodes are "cut" nodes, unless they are themselves cycle nodes. They store their role. They keep track of their cycle neighbors, so that they know when they loose the role.

"inner": Inner nodes v store that they have this role. They also know $J \cap N(v)$. Both the set of inner nodes and J are monotonically increasing. Therefore, whenever a node becomes an inner node, it can locally check whether it can join J, and inform its neighbors when it does.

"undecided": All nodes that do not fall into one of the categories above have this role. Initially, this applies to all nodes.

An important operation is the automatic maintenance of the inner nodes. Whenever a node becomes an inner node (but after the cycle and cut nodes take their roles), it will check whether it can join J. It broadcasts its decision to all of its neighbors. Neighbors that are undecided now become inner nodes too, possibly join J, and broadcast their decision. Initially, the algorithm essentially places a few inner nodes inside the FGDs. These are the seeds and independent nodes of flowers. Afterwards, it just has to maintain the cycles and ensure their feasibility, and all other nodes grow and join the sets I_f automatically. Whenever an augmenting cycle is applied, the inner nodes will "flow" into the newly discovered area.

The basic principle is that the algorithm constructs an initial FGD, i.e., a set of small FGDs. It then applies augmenting cycles wherever that results in additional inner nodes.

3.3.6 Flower-Finding Algorithm

A flower is a local structure, so checking for them is very simple. See Algorithm 3.2. The central insight here is that every node in a flower is in the $(3+\lceil L/2 \rceil)$-neighborhood of its seed, where L is the maximum length of a petal, i.e., $\max_{j=1,\dots,m} \ell_j$ in the notation of Definition 3.18. In our heuristic, we limit the petal paths to have length at most 6, hence we search for flowers only in 6-neighborhoods. The algorithm works as follows: Every node $v_0 \in V$ attempts to become some flower's seed. For that matter, it collects the subgraph on $N_6(v_0)$. This can be done in $O(\Delta^6)$ communication rounds in the *CONGEST* model.

Algorithm 3.2: Finding Flowers

```
1 Collect the subgraph on N_6(v)
2 if v is seed of a flower then
3 |   Announce update
4 |   if there are no conflicts, or v wins conflict resolve then
5 |   |   Manifest the flower
```

Then, v_0 decides for itself whether it is the seed of a flower. This does not involve any communication. Because the size of an independent set in $N_1(v_0)$ is bounded by a constant, the total number of subgraphs that need to be checked is polynomially bounded in Δ. So v_0 can enumerate all potential subgraphs.

If v_0 finds a flower, it tries to construct a local FGD for it. It informs all nodes that participate in the flower about their future role. Each node confirms that neither itself nor any of its neighbors is becoming part of a different flower, and sends a confirmation back to v_0. In case of a conflict, any tie-breaking rule is applied; for example, the seed with higher node ID wins.

If v_0 receives the confirmation, it manifests the flower. First all nodes on the boundary take their role. They store the number of independent nodes used in the flower as initial mass $M(v_0)$. They broadcast a message to their neighbors, so cut nodes also know their role.

Then, the independent nodes become inner nodes and join J. Now, the automatic extension process of inner nodes fills the remainder of the flower.

3.3.7 Augmenting Algorithm

We use a method that will search for an augmenting cycle that will lead to another FGD with a larger number of inside nodes, thereby performing one improvement step. The method is described for a single node $v_1 \in C$ that searches for an augmenting cycle containing itself. This node is the *initiator* of the search.

It runs in the following phases:

Cycle search: The node v_1 initiates the search by passing around a token. It begins with the token $T = (v_1)$. Each node that receives this token adds itself to the end of it and forwards it to a neighbor. When the token returns from there, the node forwards it to the next feasible neighbor. If there are no more neighbors, the node removes itself from the list end and returns the token to its predecessor.

The feasible neighbors to which T gets forwarded are all nodes in $V \setminus \cup_{f \in F} I_f$. The only node that may appear twice in the token is v_1, which starts the "check

solution" phase upon reception of the token. In addition, T must not contain a cycle node between two cycle neighbors. The token is limited to contain

$$|T| < \min\{\mathrm{enc}_d(M(f)) : f \in F, \exists C \in \mathscr{C}_f : T \cap C \neq \emptyset\} \quad (3.21)$$
$$\leqslant K \quad (3.22)$$

nodes. This phase can be implemented such that no node (except for v_1) has to store any information about the search. When this phase terminates unsuccessfully, i.e., without an identified augmenting cycle, the initiator exits the algorithm.

Check Solution: When the token gets forwarded to v_1, it describes a cycle. Then, v_1 sends a backtrack message backwards along T.

Backtrack: While the token travels backwards, each node performs the following: If it is a cycle node, it broadcasts a query containing T to its neighbors, which in turn respond whether they would become inside nodes after the update. Such nodes are called *new inners*. Then, the cycle node stores the number of positive responses in the token.

A non-cycle node checks whether it would have any chords after the update. In that case, it cancels the backtrack phase and informs v_1 to continue the cycle search phase.

Query Feasibility: When the backtrack message reaches v_1, feasibility is partially checked by previous steps. Now, v_1 checks the remaining conditions. Let $\mathscr{F} := \{f \in F : \exists C \in \mathscr{C}_f \text{ with } T \cap C \neq \emptyset\}$. First, it confirms that for every $f \in \mathscr{F}$, there is a matching cycle node in the token that has a nonzero new inner count. Then it picks a $f' \in \mathscr{F}$. All new inners of cycle nodes of this FGD then explore the new inner region that will exist after the update. This can be done by a BFS that carries the token. The nodes report back to v_1 the FGDs that could be reached. If this reported set equals $\mathscr{F}$, T is a feasible candidate for an update and phase "announce update" begins. Otherwise, the cycle search phase continues.

Announce update: Now T contains a feasible augmenting cycle. The initiator v_1 informs all involved nodes that an update is coming up. These nodes are T, $N(T)$, and all nodes that can be reached from any new inner in the new region. This is done by a distributed BFS as in the "query feasibility" phase.

If any node receives multiple update announcements, the initiator node of higher ID wins. The loser is then informed that its announcement failed.

Update: When the announcement successfully reached all nodes without losing a tie-break somewhere, the update is performed. If there is just one FGD involved, i.e., $|\mathscr{F}| = 1$, the update can take place immediately.

If $|\mathscr{F}| > 1$, there might be problems keeping $M(f)$ accurate if multiple augmentations happen simultaneously. So v_1 first decides that the new ID of the merged component will be v_1. It then determines what value $M(v_1)$ will have after the update. If this value strictly exceeds $\text{fit}_d(K)$, the new FGD will be large and hence independent of potential other updates; the update can take place immediately. Otherwise, the concurrent updates have to be scheduled. So v_1 floods the involved components with an update announcement, and performs its update after all others of higher priority, i.e., higher initiator ID.

Finally, all nodes in T flood their $K/2$-hop neighborhood so that cycle nodes whose cycle search phase was unsuccessful can start a new attempt, because their search space has changed.

The following important lemma follows directly from the definition for augmenting cycles in Section 3.3.4 and above algorithm description:

Lemma 3.22. *If the augmenting cycle algorithm performs an update on an FGD, it produces another FGD with strictly more inner nodes.*

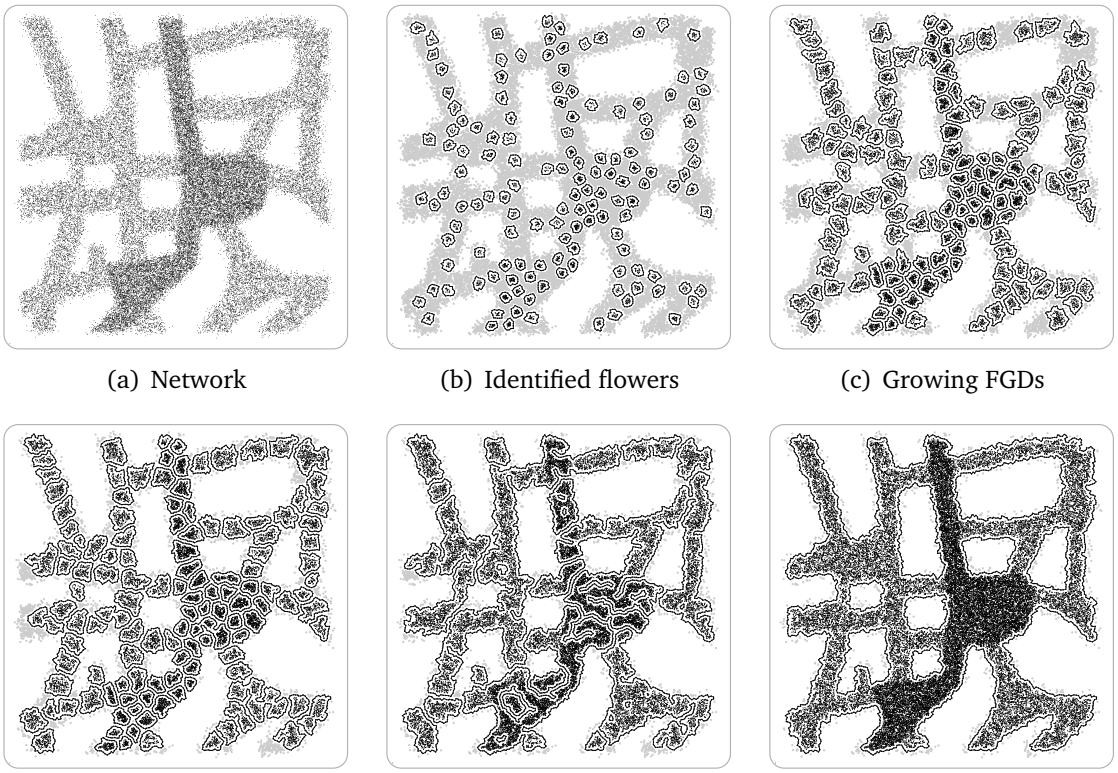
Finally, to complete the description, we give an estimate on the runtime.

Lemma 3.23. *One iteration of the augmenting cycle algorithm for a given initiator node has message complexity $O(\Delta_K^K n)$ and time complexity $O(\Delta_K^K \Delta_1 + n)$.*

Proof. There are at most Δ_K^K cycles that are checked. For one cycle, the backtrack phase takes $O(\Delta_1)$ message and time complexity. The query feasibility phase involves flooding the part of the new inside that is contained in the cycle. Because there can be any number of nodes in this region, message complexity for this flood is $O(n)$. The flood will be finished after at most $2\text{fit}_d(K)$ communication rounds, the time complexity is therefore $O(1)$. After a feasible cycle is found, the announce update and update phases are performed once. Both involve a constant number of floods over the network, their message and time complexities are therefore $O(n)$. Combining these complexities results in the claimed values. □

3.4 Experimental Evaluation

Our boundary detection algorithm is a heuristic. Since we cannot prove how well the quality of its output is, we have to rely on simulations. This section presents the result of the algorithm on some complex networks. Furthermore, we compare the algorithm with some of its competitors.



(a) Network (b) Identified flowers (c) Growing FGDs

(d) FGDs beginning to merge (e) Mergings FGDs (f) Final state

Figure 3.15: Simulation output for our boundary detection algorithm. *The varying density and fuzzy boundaries could not stop our algorithm from producing a perfect boundary representation.*

The first network is shown in Figure 3.15(a). It is a UDG with 50 000 nodes. The topology is from an urban scenario, where nodes were scattered in the streets. This particular network is also featured in our video [FK06a] that showcases the boundary detection algorithm as well as a preliminary version of the clustering scheme discussed in Chapter 4. The network has two important features:

- There is no uniform density. The vertical street in the middle of the network has an average degree of 30, while the remaining parts have just 20. This is supposed to show how our algorithm is immune to such density fluctuations, which pose significant challenges to stochastic algorithms: It distorts the histograms, and it adds a fake boundary between the high- and the low-density areas.

- There is no sharp boundary, i.e., the density does not drop sharply to zero when leaving the street area. Again, this is to show another benefit of our

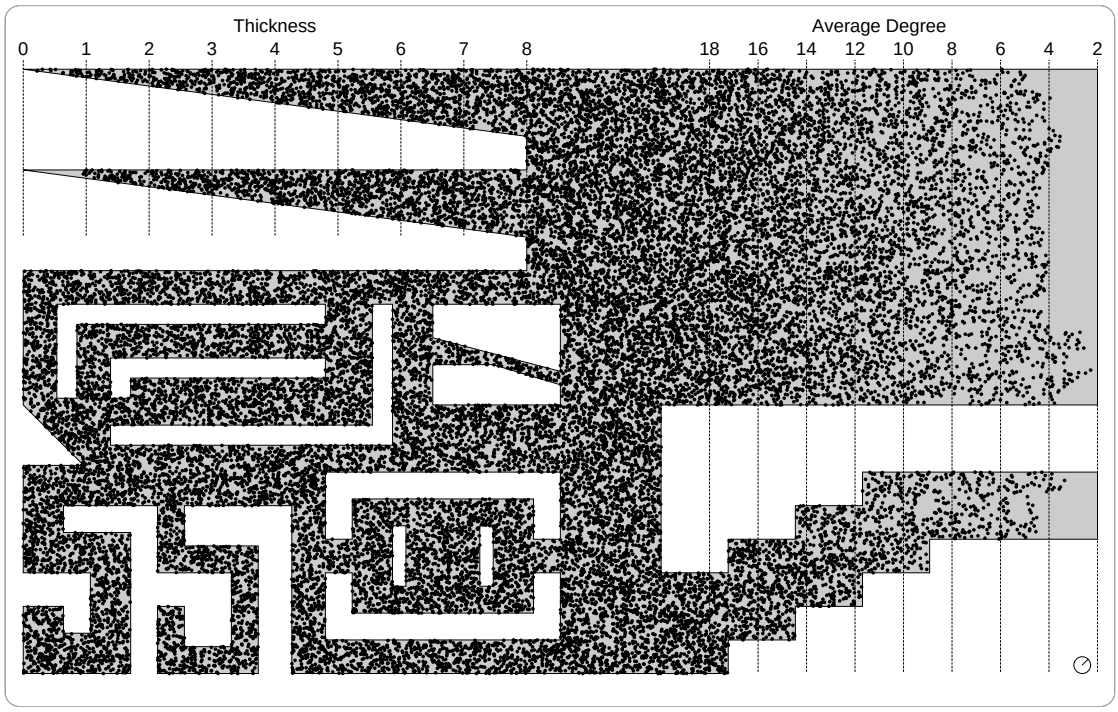


Figure 3.16: Network instance for boundary detection tests. *Unit Disk Graph (range shown in the lower right corner) with 31 222 nodes.*

algorithm over stochastic ones, as this sharp drop is what the previously shown algorithm (Section 3.2) attempts to find.

The output and several intermediate FGDs are shown in Figure 3.15. It is easy to see how the initially found flowers (3.15(b)) grow until they touch (3.15(c) and 3.15(d)), then start to merge together (3.15(e)) until finally a single FGD emerges, which perfectly describes the actual boundary (3.15(f)).

3.4.1 Complex Network Setup

To compare different algorithms, we used another network. It is shown in Figure 3.16. It consists of 31 222 nodes, and the graph is a UDG (the communication range is shown in the lower right corner). We tried to include both "easy" and "hard to tackle" areas in the network, so that every algorithm would fail in certain areas, but excel in others. The relevant features of this instance are:

Varying Degree: In most of the network, the node distribution is uniform with an expected degree of 20. We chose 20 because the papers whose algorithms we test in Section 3.4.3 claim that they can work with densities strictly less than 20.

In the right part, the density fades out. The distribution's density function is affinely decreasing in the +X direction. The vertical lines marked "18" to "2" describe the expected neighborhood size of a node on these lines (the expected size from the density only, ignoring effects from the polygonal boundary). This area's purpose is to show how sparse networks can become before the algorithms fail or produce erratical results.

The sudden end of the network around density 4 is because the network consists of just one connected component. We first populated the network with the shown densities, but then only kept this component. This is because in a distributed, static network, having multiple components is equivalent to having multiple independent networks. So the additional components have no value, but may add to confusion.

Thin Spikes: In the top left area, there are two spikes protruding to the left. The lines marked "0" to "8" show the thickness of the spikes, measured in vertical direction. In the lower left corner, there are two twisted spikes (in the shape of a "5") with a thickness of 4.8 resp. 3.2.

The purpose of the spikes is twofold: First, we want to see how thin a corridor can be for an algorithm to still produce a well-defined boundary. Second, how thin can it be until there are no more flowers in it?

Maze: The remainder of the left part is filled with convex and concave obstacles, forming a complex topology. There are corridors of different width, and holes whose shortest enclosing cycle touches other holes.

3.4.2 Our algorithm

First, we report on the result from our algorithm. Figure 3.17 shows the flowers that were detected. It can be seen that throughout the high-density area, flowers are found wherever the available space permits it. Note that there are even more flowers in the graph, but the requirement of being disjoint prevents them from being in the solution. Figure 3.18 shows the final result. We draw the following conclusions:

- The minimum corridor width is about 4. This can be seen in the thinning spikes, as well as from the fact that the bottom left spike of width 4.8 is perfectly detected, while the algorithm was not able to enter the 3.2 spike at all.

- The density can be as low as 12. Even in the density region 12–10, the detected boundary is clean with just one fake hole.

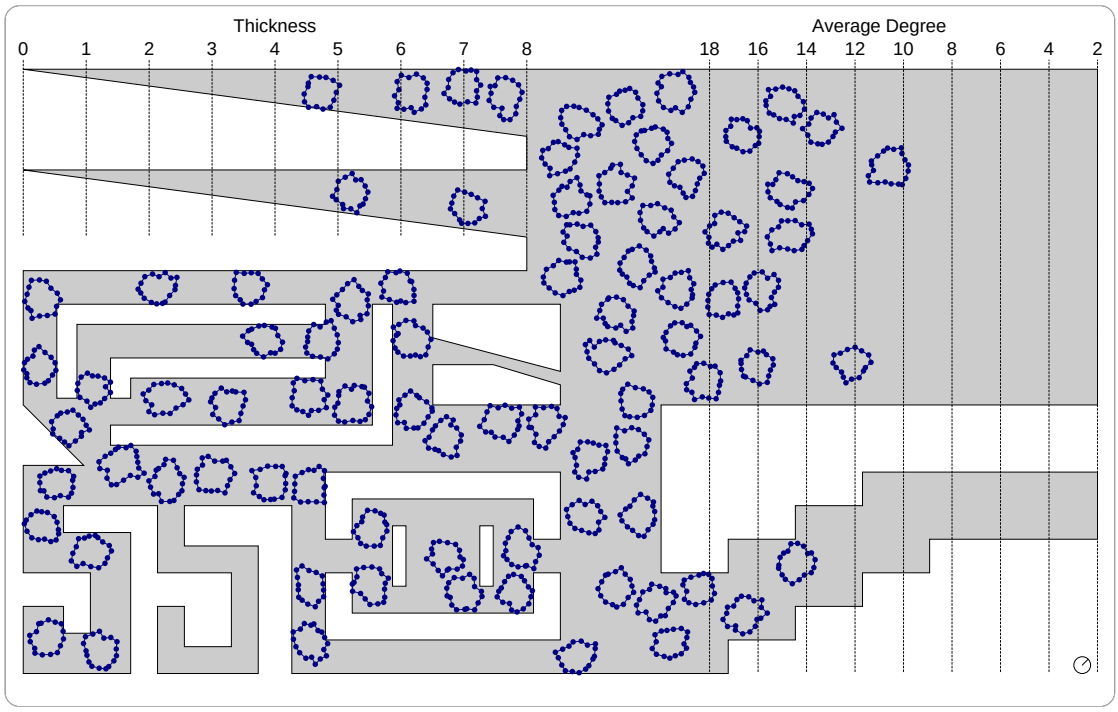


Figure 3.17: Output of flower identification. *83 disjoint flowers were identified. Note that a single one would have been sufficient to enter the augmentation stage.*

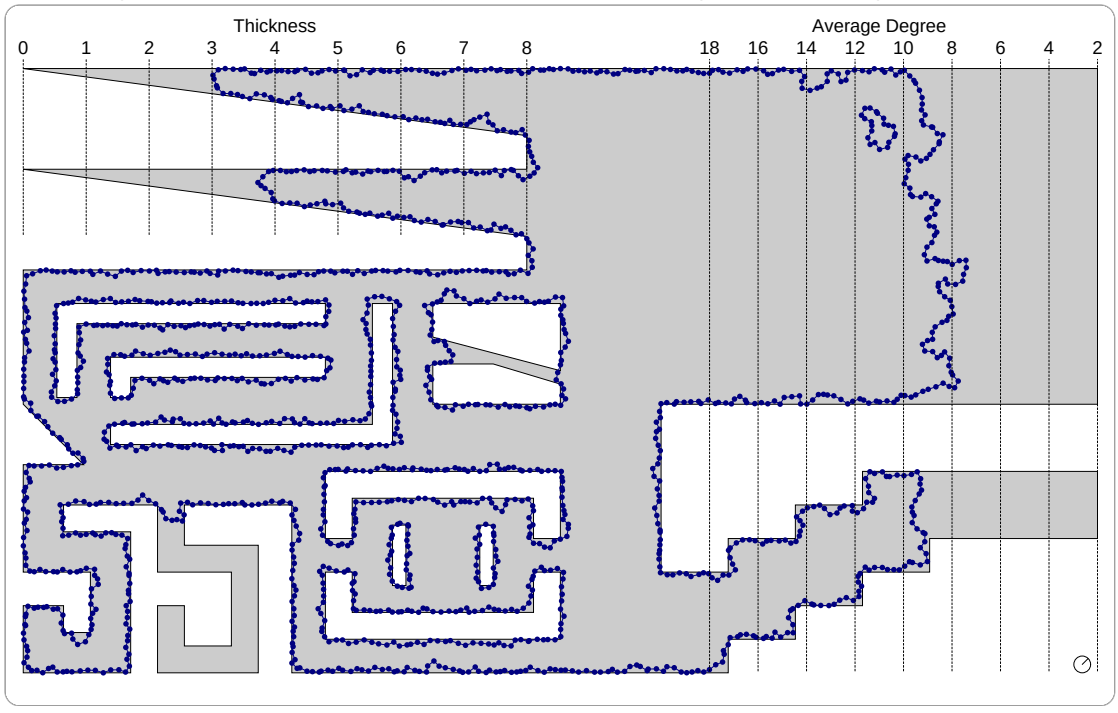


Figure 3.18: Final result of our boundary detection algorithm. *Thin corridors and densities below 12 are problematic. Other than that, the detected boundary is seemingly perfect.*

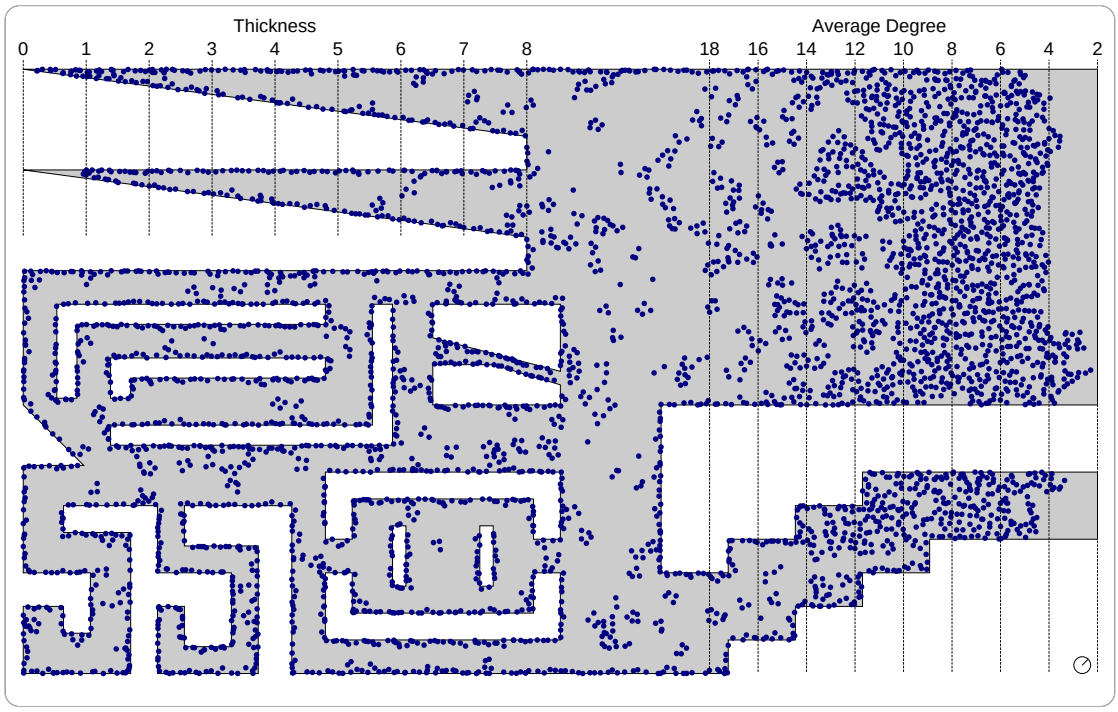


Figure 3.19: Output of the Martincic and Schwiebert algorithm [MS04]. *In this algorithm, every node v has full position information on $N_2(v)$.*

- The complex maze poses no problem at all, the algorithm perfectly identified all holes and their boundaries, with the exception of the small corridor between two holes, which got glued into one.

3.4.3 Compare with Others

To demonstrate how well our algorithm performs, we accompany it with comparative results using similar algorithms. In this section, we discuss two competing algorithms in the context of our example network, see Section 3.1.1 for more information on these algorithms.

Martincic and Schwiebert: This algorithm has full location knowledge available, without any estimation errors. It therefore belongs to a different league. We included it here to have a comparison and rationale on how much easier the problem becomes if localization is available. See Figure 3.19 for the result. The output is surprisingly bad. While there are marked nodes everywhere close to the boundary, the many false positives show that $N_2(v)$ does not contain enough information to decide whether v is a boundary node.

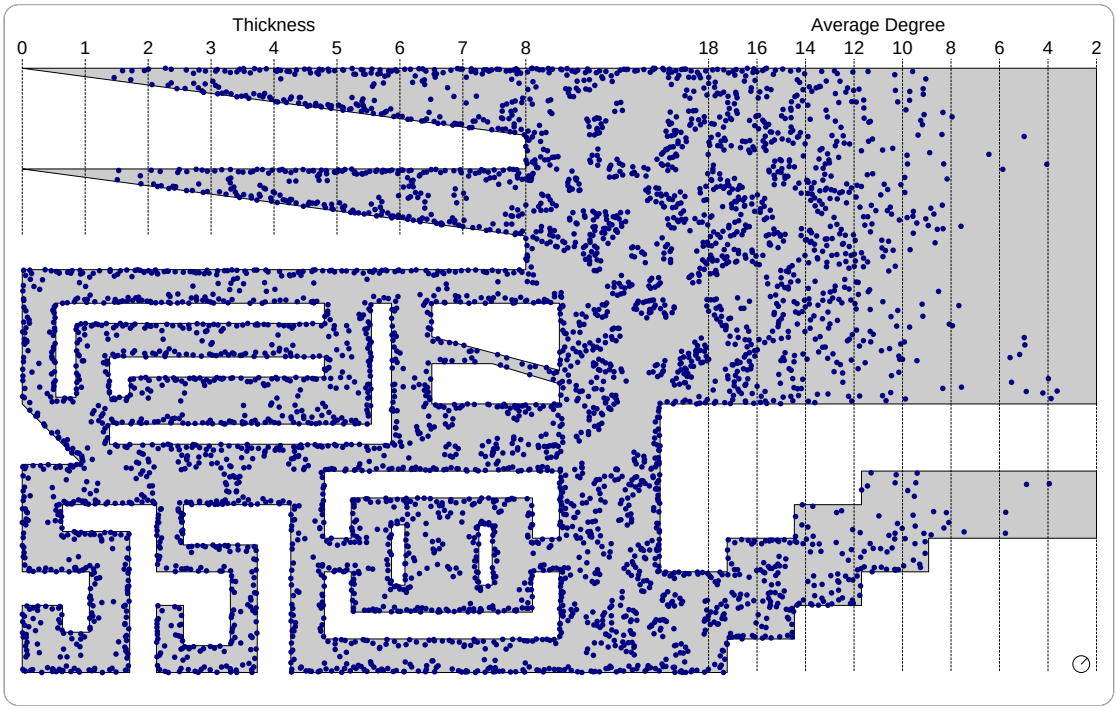


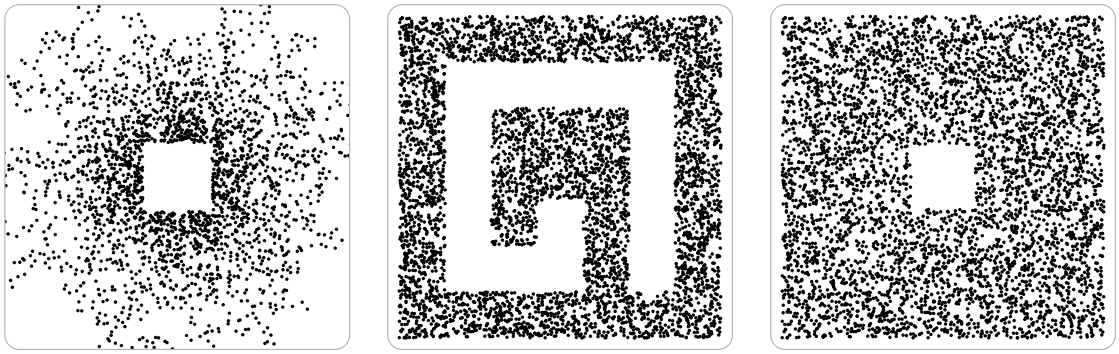
Figure 3.20: Output of the Funke and Klein algorithm [FK06b]. *This algorithm has no problem with narrow corridors, but produces many false positives.*

Funke and Klein: The output of this algorithm can be found in Figure 3.20. There is a clear concentration of correctly identified nodes at the network boundary. Again, there are many false positives on the insides. Apparently, the density that is required for this algorithm is higher than 20; the examples in the original paper show successful runs in such networks.

An important point here is, that this algorithm does not provide any ordering on the detected boundary. Connecting the marked nodes to paths or cycles is supposed to happen in a post-processing step. This seems nearly impossible on the data that was produced by the algorithm.

Wang, Gao, and Mitchell: This is the second algorithm that uses the same underlying assumptions as ours, so we would have loved to include the result here. Unfortunately, this turned out to be impossible. There exists a prototype implementation that was used by the original authors to conduct the experimental analysis in [WGM06], with promising results. The authors were even kind enough to make this implementation available to us. However, it turned out that it relies heavily on matrices that allow fast running time on small networks, but would consume 21 GByte of RAM for our scenario.

Therefore, we had to adopt the implementation and rewrite large parts of it.



(a) 2558 nodes (b) 5000 nodes (c) 5000 nodes

Figure 3.21: Smaller Networks. *The left network has an interesting variation in density, while the other two are supposedly easy to solve.*

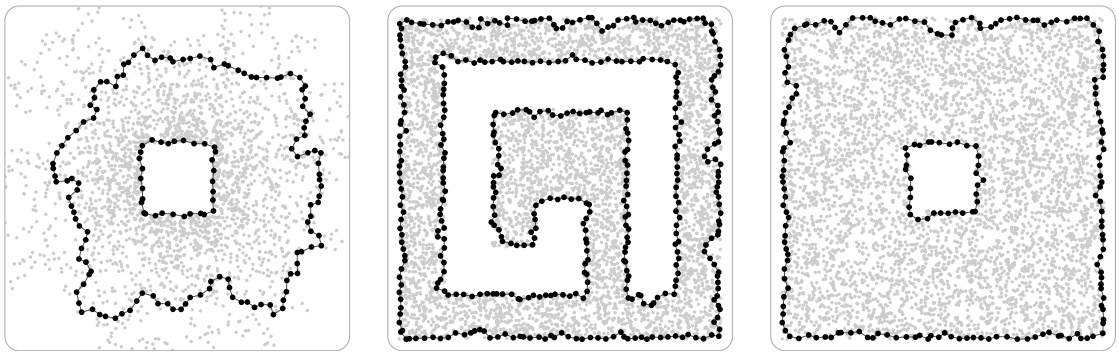
Running this code on the network resulted in a mess. The initial cycle that defines the order of extremal nodes was very small, enclosing a tiny hole in the low-density region in the top right corner. This caused the extremal nodes to have very similar positions, and the order in which they had to be connected was seemingly random. This confirmed the major weakness of the algorithm: If there is a complex structure far away from the initial cycle, the method of following extremal nodes no longer works.

Independent on whether there are additional tests dealing with this case in the original implementation, we concluded that trying to come up with a way to resolve such problems meant extending and improving the proposed algorithm. This would have lead us far from a simple comparison, into the area of developing new algorithms. Hence, we decided not to include this algorithm's result. Instead, we ran another set of comparisons on smaller data.

3.4.4 Further Comparison with Others

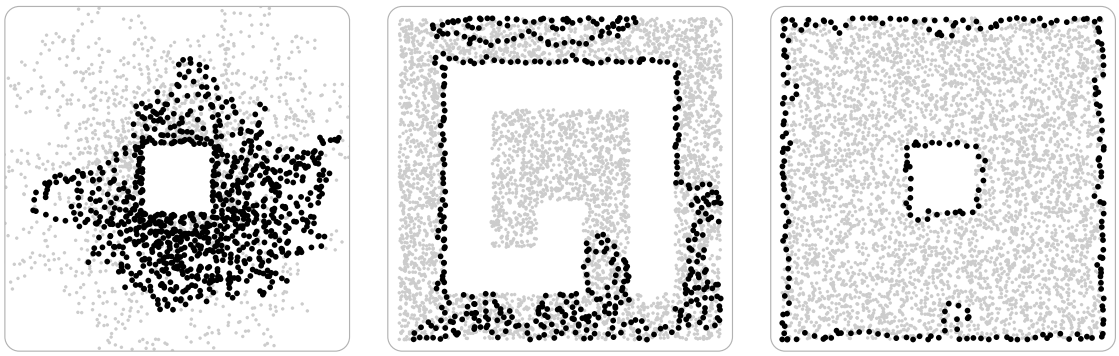
Next, we constructed three smaller networks, see Figure 3.21. The first network consists of one connected component, where the distance from the center follows an exponential distribution (neglecting the hole). It has two completely different boundaries: The perfect rectangle in the middle, with high-degree nodes close to it. The degree is over 40 for most of the nodes there. The second boundary is hard to define, as the density is continuously degrading towards zero. The other two networks have a uniform distribution, both with an average degree of 17, see Figure 3.21(b). The low density is the only challenge in them.

Our algorithm's output is shown in Figure 3.22. In all three cases, it detected the correct number of boundaries and produced clean cycles for them.



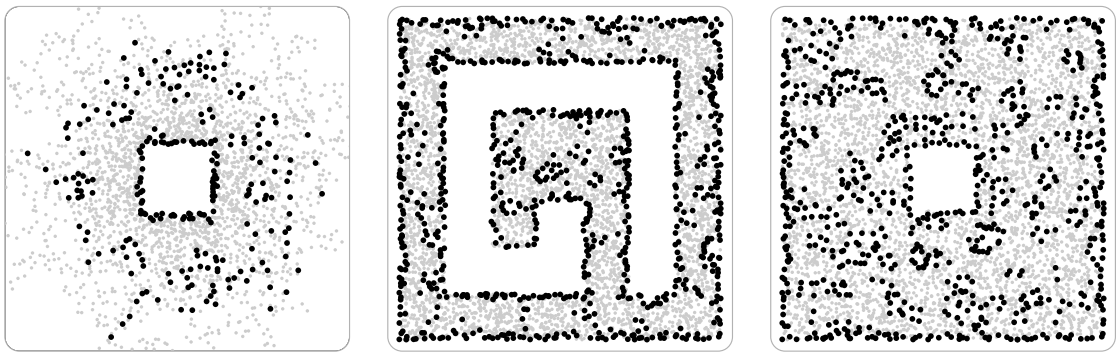
(a) (b) (c)

Figure 3.22: Our Algorithm. *The two boundaries in each network are identified. The algorithm's ability to deal with a fading density clearly helped for (a).*



(a) (b) (c)

Figure 3.23: Wang, Gao, and Mitchell. *This algorithm runs into problems whenever the initial cycle is in an unfortunate position, as seen in (a) and (b). The boundary in (c) is perfectly detected.*



(a) (b) (c)

Figure 3.24: Funke and Klein. *There is a visible concentration of marked nodes close to the boundaries. However, the algorithm cannot cope with low densities.*

For the algorithm by Wang, Gao, and Mitchell, we could use the original implementation, see Figure 3.23. In the first network, the initial cycle that determines the order of extremal nodes was small, embracing some hole in the low-density area. This causes the boundary cycles to become seemingly random. The third network posed no problem, the result is flawless. We fail to provide an explanation for the outcome in the second one though.

Figure 3.24 shows results from the Funke and Klein algorithm. As with the large example in the previous section, the boundary is identified, but there are too many marked nodes on the inside.

3.4.5 Conclusions

We gained the following central insight from the simulations: None of the algorithms can really detect the boundary ∂A in a clean and concise manner. This was particularly surprising for the algorithm with full location information.

This shows a big advantage of our algorithm, convincing us that it is indeed by far more useful than all of the competition: The algorithm is aware of the fact that it will not work perfectly. Instead, it will identify a part of the network and identify the boundary of that part. It clearly marks where it was not able to succeed by leaving the nodes there in the "undecided" state. For the remaining regions, there is the proof that the produced boundary cycles actually have a precisely defined geometric interpretation.

Chapter 4

Clustering

In this chapter, we utilize the boundary detection heuristics from the previous chapter to construct a new form of location information, based on clusters. We build a segmentation that reflects the network topology, and describe how to benefit from it in applications.

4.1 Shape Segmentation

First, we look at a shape segmentation problem. As with boundary detection, there is no actual definition of it. Informally, the target is to find a segmentation of the network area into pieces with disjoint interiors, that is, sets $A_1, \ldots, A_k$ such that

$$A_1 \cup \ldots \cup A_k = A \text{ and } A_i^\circ \cap A_j^\circ = \varnothing \; \forall i \neq j \; . \tag{4.1}$$

Obviously, it is simple to find a segmentation unless there are additional constraints. The challenge is to find a segmentation that

1. can be computed by a distributed sensor network and
2. has provable beneficial properties for the network.

As before, we raise the bar by also requiring that

3. the segmentation must be computed without coordinates.

The second constraint allows for a broad range of segmentation types. A "provable property" will be beneficial generally just for certain applications. For example, there is a huge amount of literature on clustering a network such that each cluster has a cluster-head, which is responsible to coordinate the communication of the other cluster members (all of which are usually direct neighbors of the head), with the intention of reducing interference [Pel00, WW07]. Such a clustering tells nothing about the network geometry though.

Here, we present a segmentation scheme that follows a "street map" motivation: We identify disjoint, convex areas analogous to intersections, tunnel-shaped areas

that connect intersections, and dead-ends. That is, the clustering reflects the topological structure of the network. It will be used in Section 4.2 to actually compute the network analogy of a street map, enabling the sensor nodes to get a good picture of the global network structure and their own role in it, without using classic localization.

Related Work: There have not been many publications on this topic so far. The first paper on this topic was [KFPF06], where we proposed the deterministic boundary algorithm from Chapter 3 and a preliminary version of the clustering scheme we are presenting here. It was also used in our video on the subject [FK06a].

Another heuristic was proposed by Zhu, Sarkar, and Gao [ZSG07]. Their approach is centered around medial vertices, as is both our previous and current one. They identify a subset of medial vertices to become cores of clusters, called sinks. The remaining network gets clustered using a flow complex that spans A. Each node joins the sink to which its local flow is directed. They demonstrate a well-shaped clustering using simulations. There are no provable segmentation properties.

4.1.1 Continuous Case

We define and analyze our segmentation scheme using a purely geometric approach. We consider the continuous case, i.e., assume that every point $x \in A$ corresponds to a sensor node in an infinitely dense network. First we define the segmentation and prove said beneficial properties for the continuous shape. Afterwards, we present discrete analogons and distributed algorithms to compute them, and show that the outcome is sufficiently close to the continuous segmentation by running experiments.

We assume the network's boundary ∂A to consist of straight lines and circular arcs. Then, the medial axis is as described in Section 2.2.5. We use the medial axis to define our clustering scheme.

Definition 4.1. *In the continuous setting, our segmentation is defined as follows:*

1. *For every $x \in V_3(A)$,* $\operatorname{conv}(C(x))$ *defines a cluster. These are called* vertex clusters.

2. *The closures of the connected components of the remaining area*

$$A \setminus \bigcup_{x \in V_3(A)} \operatorname{conv}(C(x))$$

become tunnel clusters.

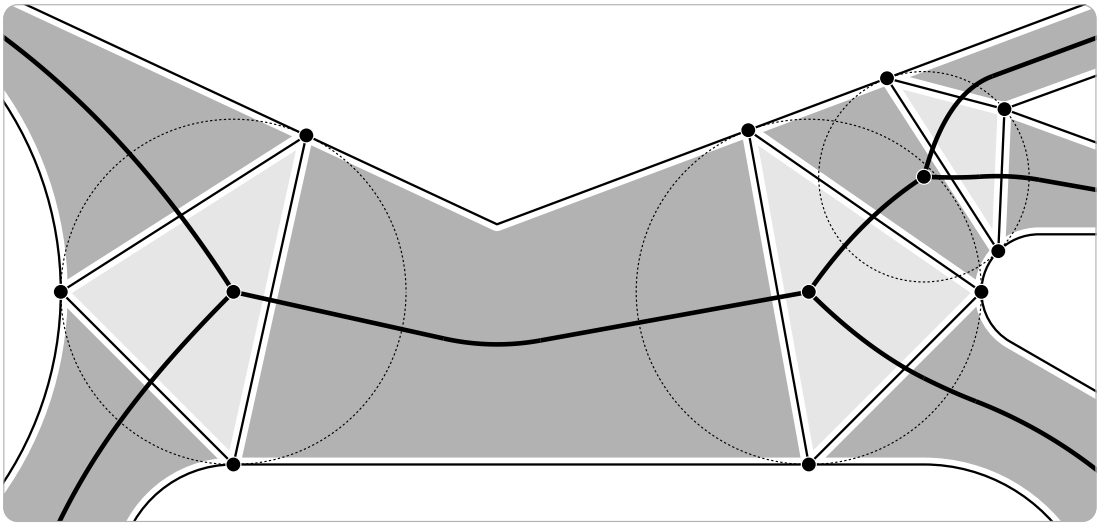


Figure 4.1: Continuous-case segmentation. *Based on the three medial vertices, three vertex clusters arise. Also, there are two completely visible and five cropped tunnel clusters.*

See Figure 4.1 for an example. Note that the medial vertex defining a cluster is not always part of it, as shown for the right-most vertex cluster in the figure. In the following we analyze and prove some properties of these clusters. The first two are trivial:

Observation 4.2. *Vertex clusters are convex. The number of vertex clusters is finite.*

Proof. Convexity follows from the definition, and finiteness from the fact that the medial axis is a finite graph; see Section 2.2.5. □

In addition to these properties, we can prove that vertex clusters have a particular shape:

Lemma 4.3. *The boundary of a vertex cluster* $\mathrm{conv}(C(x))$ *is a finite alternating sequence of*

1. *Contact components of* x *and*
2. *Chords of the circle* $\partial B_r(x)$*, where* $(x, r) \in \mathbf{MAT}(A)$*.*

We call the latter kind the *exits* of cluster $\mathrm{conv}(C(x))$.

Proof. The contact components are closed sets, as they are components of the intersection of two closed sets, ∂A and $\partial B_r(x)$. Sort the components by their order on the boundary. Assume y is the last point on one component, and z is the first on the following one. Then y and z define a tight half-space that includes $\mathrm{conv}(C(x))$, hence the line yz is part of $\partial\,\mathrm{conv}(C(x))$, and it is a chord of the circle.

The sequence is finite because the number of contact components is finite under our assumptions on the boundary structure [CCM97]. □

To show that our clustering scheme is actually a segmentation as introduced at the beginning of this chapter, a final property needs to be shown:

Lemma 4.4. *The vertex clusters have disjoint interiors.*

Proof. Let $\text{conv}(C(x_1))$ and $\text{conv}(C(x_2))$ be two vertex clusters, with radii r_1 and r_2. If $B_{r_1}(x_1)$ and $B_{r_2}(x_2)$ are disjoint, then the clusters are also disjoint. They cannot contain each other, as both have points from ∂A on the circle, but no such points on the inside. If they touch, their interiors are disjoint. Hence, the only critical case is when they intersect.

In this case, there are exactly two circle intersection points, so let $\{y,z\} = \partial B_{r_1}(x_1) \cap \partial B_{r_2}(x_2)$. These points cut both circles into two pieces, where each circle contains one piece of the other circle on the inside. No contact point of x_1 is in $B_{r_2}(x_2) \setminus \{y,z\}$, and vice versa. So y and z define two half-spaces, each of which contains all contact components for one cluster. This limits the intersection of the clusters to the line yz, which cannot contain interior points of the clusters. □

This concludes our discussion on the vertex clusters, and we turn to the other kind. Again, it can be shown that these have a simple boundary structure:

Lemma 4.5. *Let T be a tunnel cluster. Then one of the following holds:*

1. *Either $T = A$, or*
2. *∂T consists of one exit of some vertex cluster and one continuous piece of a boundary curve, or*
3. *∂T consists of two vertex cluster exits (possibly two exits of the same cluster) and two continuous pieces of boundary curves.*

Proof. Assume $T \neq A$. Because A° is connected, T shares a point with some vertex cluster $\text{conv}(C(x_0))$ with radius r_0. By Lemma 4.3, the intersection is a complete chord a_0b_0 of $\partial B_{r_0}(x_0)$.

Intuitively, the idea is now to squeeze the circle $\partial B_{r_0}(x_0)$ between a_0 and b_0 through the tunnel and see where it stops. Given a carefully chosen parameterization [CCM97] of the curves that make up $\mathbf{MA}(A)$, as well a parameterization of ∂A, one can follow the unique edge leaving x_0 that belongs to a_0 and b_0. Let these curves be $x(t), a(t), b(t)$, $t \in [0,1]$, such that $x(0) = x_0$, $a(0) = a_0$ and $b(0) = b_0$, with the following properties:

- $x(1)$ is the endpoint of the geometric edge, i.e., either a leaf, or a medial vertex.

- There is a radius function r such that $(x(t), r(t)) \in \mathbf{MAT}(A)\ \forall t \in [0,1]$.
- For all $t \in [0,1]$, $a(t)$ and $b(t)$ are contact points of $x(t)$.
- a and b are monotonic parameterizations of two pieces of ∂A.

These curves are easily obtained [CCM97], directly following the geometric intuition. At $x(1)$, there are two possibilities: First, there could be a leaf of the medial axis, in which case $a(1)$ and $b(1)$ are in the only contact component of $x(1)$. In this case, the outer boundary of T follows the second option of this lemma's claim.

Second, $x(1)$ may be a medial vertex. In this case, $a(1)$ and $b(1)$ are on different contact components of $x(1)$, and there is chord of $\partial B_{r(1)}(x(1))$ connecting these components. By shortening or elongating a and b to meet the chord, the structure of the last option in the claim is obtained.

The final missing piece is that T contains no holes, i.e., no inner boundaries. This follows from the fact that the line $a(t)b(t)$ is a chord of $\partial B_{r(t)}(x(t))\ \forall t \in [0,1]$, hence fully contained in A°, with the exception of the endpoints. Therefore, this line defines a sweep between the ends (if $x(1)$ is a medial vertex) resp. the entrance and final circle (if $x(1)$ is a leaf) that does not touch additional boundaries. If $x(1)$ is a leaf and $B^\circ_{r(1)}(x(1)) \neq \emptyset$, this circle covers the remainder of T. □

4.1.2 Discrete Case

Now that we defined our clustering scheme in the continuous case, we describe a heuristic approach to mimic it in the discrete case without location information. Essentially, we use boundary cycles of an FGD instead of ∂A, and replace all geometric distances with hop counts. We assume that there is a boundary description which is feasible for the complete network. This could be the boundary cycles from the algorithm in Chapter 3, by removing all nodes that were left in the "undecided" state. It can also be any similar structure, given that all boundaries are closed cycles in the network with an orientation, i.e., every boundary node knows it predecessor and successor on the boundary to which it belongs.

Let these cycles be $\mathscr{C} \subset 2^V$. We assume that each cycle $C \in \mathscr{C}$ is numbered, i.e.,

$$C = (v_1(C), , \ldots, v_{|C|}(C)) \quad \forall C \in \mathscr{C}\ . \tag{4.2}$$

We use a measure $\tilde{d}$ that describes the distance of boundary nodes. For two nodes $v_j(C)$ and $v_{j'}(C')$, we define

$$\tilde{d}(v_j(C), v_{j'}(C')) := \begin{cases} +\infty & \text{if } C \neq C' \\ \min\{|j'-j|, |C| - |j'-j|\} & \text{if } C = C'\ . \end{cases} \tag{4.3}$$

That is, $\tilde{d}$ assigns nodes on the same boundary their distance within this boundary, and ∞ to nodes on different boundaries. For the discrete analogon of the medial axis, we need to reduce the geometrically defined distances to hop-based ones. This is provided by the following definition:

Definition 4.6. *Let $v \in V$. We denote by*

- $r(v, C) := \min\{r : N_r(v) \cap C \neq \emptyset\}$ *the distance from v to the boundary $C \in \mathscr{C}$,*
- $r(v) := \min\{r(v, C) : C \in \mathscr{C}\}$ *the minimal distance to a boundary node,*
- $Q(v, C) := N_{r(v,C)} \cap C$ *the boundary nodes of $C \in \mathscr{C}$ at minimum distance, and*
- $Q(v) := N_{r(v)} \cap (\cup_{C \in \mathscr{C}})$.

The nodes in $Q(v)$ are called contact nodes.

This allows to define nodes that take the role of medial vertices.

Definition 4.7. *A node $v_1 \in V$ is called a k-*medial node*, if there are $k-1$ nodes $\{v_2, \ldots, v_k\} \subseteq N_1(v_1)$ and $q_1, \ldots, q_k$, $q_i \in Q(v_i)$ $\forall i = 1, \ldots, k$, such that $\tilde{d}(q_i, q_j) > (r(v)+1)\pi$. The nodes $v_1, \ldots, v_k$ do not have to be different nodes. V_k denotes the set of all k-medial nodes in G.*

A medial vertex is a single point in the geometric case, but the inaccuracy in hop-distances will make many nodes claim to be the corresponding medial node. We simply build groups of them. The nodes belonging to the same medial vertex will be close to each other, but sometimes not close enough to be connected. Therefore, we use connected components over 2 hops:

Definition 4.8. *Let $G' = (V_3, E^2)$ be the graph on V_3, where two nodes u and v are adjacent in G' if $u \in N_2(v)$ in G. The connected components of G' are called* vertex cluster cores *of G.*

Now that we have a discrete analogon of medial vertices, we can use it to define what a vertex cluster is in this setting. Note that we simply use the original definition, restated in the network terms.

Definition 4.9. *Let $R \subseteq V$ be a vertex cluster core. The set*

$$\mathrm{VC}(R) := \{v \in V : \; p(v) \in \mathrm{conv}\{p(q) : \; q \in \cup_{v \in R} Q(v)\}\} \tag{4.4}$$

is called vertex cluster *of R.*

Algorithm 4.1: Stages in distributed clustering (overview)

1 Synchronize end of boundary detection
2 Label boundaries
3 Identify 3-medial nodes
4 Build vertex clusters
5 Build tunnel clusters

4.1.3 Distributed Algorithm

After turning the continuous segmentation definition into a discrete one, we discuss a distributed heuristic to construct it. It works in five stages, see Algorithm 4.1. The stages are performed as follows:

Synchronize: This phase runs in parallel to the boundary detection algorithm. The second phase needs to be started at all cycle nodes simultaneously, after the boundary detection terminates. For that matter, we use a synchronization tree in the network, i.e., a spanning tree. Every node in the tree keeps track of whether there are any active initiator nodes in its subtree. When the synchronization tree root detects that there are no more initiators, it informs the cycle nodes to start the second phase. Because the root knows the tree depth, it can ensure the second phase starts in sync.

Label: Now the cycle nodes assign themselves consecutive numbers. When a node $v \in C$ for some $C \in \mathscr{C}$ receives the message to start this phase over the tree, it generates a token $(v, 0)$. In every round, each cycle node inspects all tokens it currently possesses, and discards all of them, with the exception of a token that has the highest ID (first entry) among all current and earlier tokens. If such a token exists, the node increases the counter (second entry) and forwards it to a cycle neighbor from which the token did not originate. Finally, when a node u receives a token it generated itself, it concludes that this must be the only surviving token in C, and the counter equals $|C|$. So u becomes $v_1(C)$, the first node on the cycle. The ID of it will be used as the ID of C. It sends the token around the cycle for a second time, so all nodes on the cycle know $|C|$, their consecutive position on C, and the ID $v_1(C)$ afterwards.

3-Medial Nodes: This phase identifies the 3-medial nodes. We use a simple heuristic, because computing $Q(v)$ for all v is too expensive. We establish that every node v obtains the following information: First, the distance $r(v)$ to the boundary. Second, for every C with $r(v, C) = r(v)$, v also receives $|C|$, $v_1(C)$, and an i for which

$v_i(C) \in Q(v)$ holds.

All cycle nodes start a BFS simultaneously. This is scheduled over the synchronization tree. Each cycle node $v_j(C)$ sends the message $(v_1(C), j, |C|, 0)$ to all of its neighbors. Every node only forwards messages (v, i, s, d) if

1. it has not seen any message with smaller distance d, and

2. it has not seen a message with same distance d from the same boundary (detected by its ID v).

When a nodes forwards the message, it increases the distance counter by 1.

When the BFS finishes, each node shares its information with its direct neighbors. Now, each node v checks whether it is a 3-medial node according to Definition 4.7. For that matter, it ignores that there may be more contact nodes than what is locally known. All nodes that conclude to be 3-medial nodes join the set $\tilde{V}_3 \subseteq V_3$.

Finally, the nodes in $\tilde{V}_3$ determine the vertex cluster cores by computing their connected components with two-hop edges, and assign each component R an unique ID number. $\mathscr{R} \subset 2^V$ denotes the set of these components.

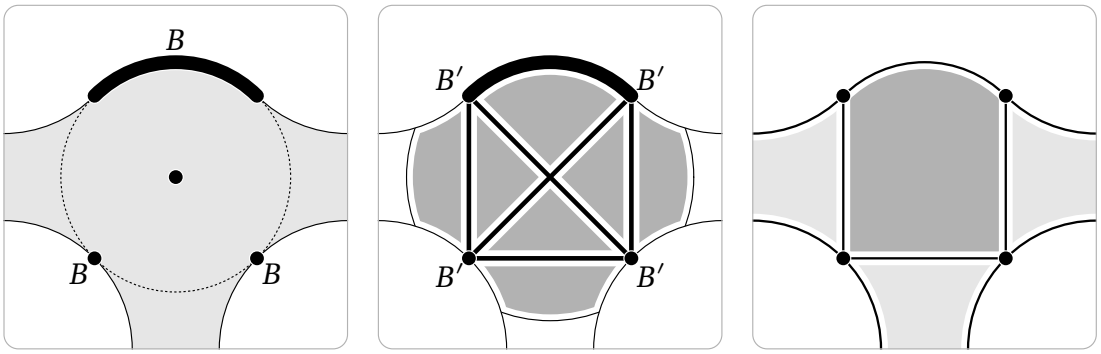
Vertex Clusters: Now, the network constructs a vertex cluster for every core R. By attaching the core's ID, the protocols for the different cores run in parallel. We focus on a single core in the algorithm's description.

Computing VC(R) requires a way to determine whether a node is located in the convex hull of some other nodes' positions. We are not aware how to do that generally, if location information is not available. However, we can exploit that the contact nodes are located on a circle.

See Figure 4.2 for a visualization of this step. First, all nodes $v \in R$ send a broadcast message to $N_{r(v)+1}$. All boundary nodes in $N_{r(v)}$ that receive such a message for a given core ID become contact nodes for R, forming the set B (Figure 4.2(a)). Some of these nodes may be consecutive nodes on the boundary: They determine the set B' of all nodes in B with less than two neighbors in B. Each node in B' starts a BFS to determine a shortest path to each other node in B' (Figure 4.2(b)). Now, we shrink the cluster using these paths: Initially, all nodes in $\cup_{v \in R} N_{r(v)+1}$ are part of the cluster. After the shortest paths are found, they are used as cuts. Consider the node set

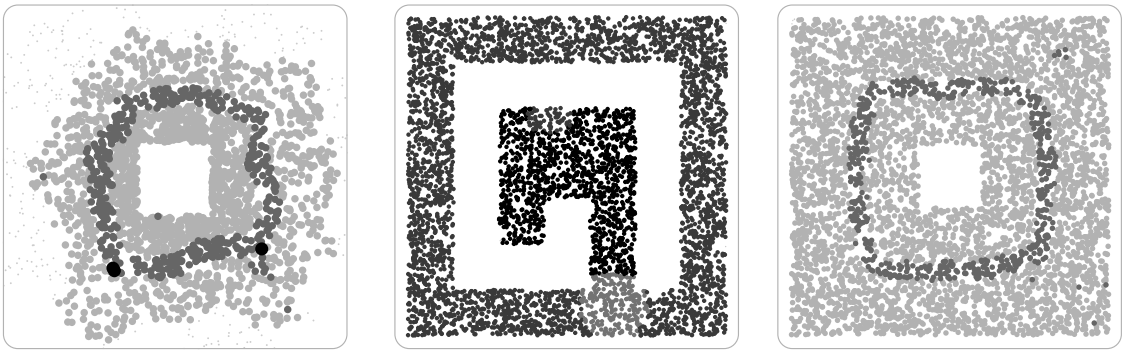
$$X := (\cup_{v \in R} N_{r(v)+1}) \setminus (\cup_{v \in R} N_{r(v)}) \,. \tag{4.5}$$

Each node $v \in X$ that is no neighbor of a shortest path node removes itself from VC(R). Then, all neighbors of former X nodes are removed, unless they are shortest path neighbors. This is repeated, until no further removals are possible. See Figure 4.2(c). The remaining nodes are the final members of VC(R).



(a) (b) (c)

Figure 4.2: Constructing conv($Q(x)$). *A vertex cluster is built from $N_{r(v)}(v)$ by enlarging it, fixing shortest paths between contact nodes, and shrinking from the extremal nodes.*



(a) V_2 nodes (b) Clustering (c) V_2 nodes

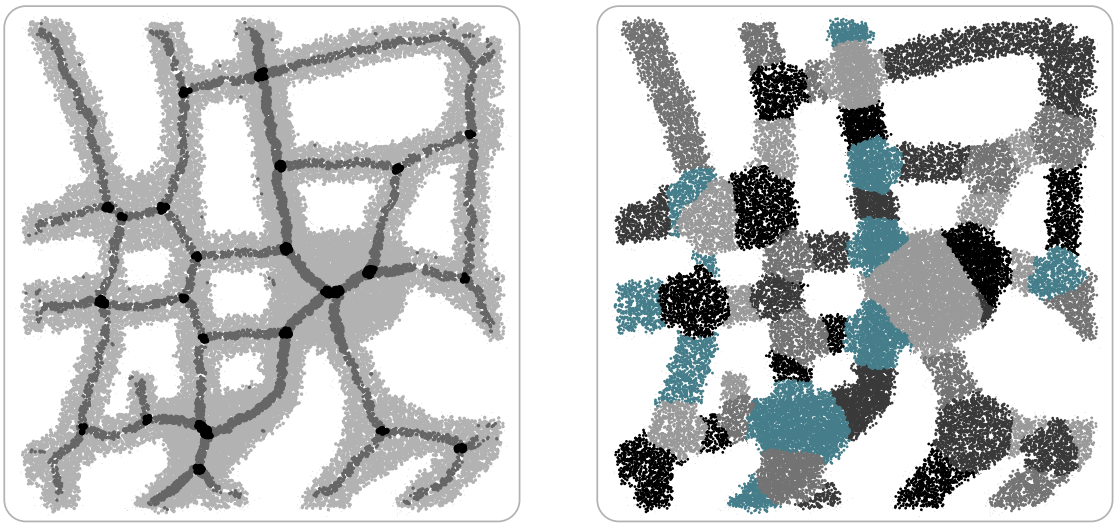
Figure 4.3: Small network results. *In (a) and (c), the topology is too simple for a clustering. (b) shows an actual segmentation.*

Tunnel Clusters: Tunnel clusters are simply the connected components of all remaining nodes. So, after all vertex clusters are built, a final connected components algorithm is triggered over the synchronization tree. The resulting components become the set $\mathcal{T} \subseteq 2^V$.

4.1.4 Example

To show that the distributed heuristic actually produces useful results, we ran simulations. We used the output of the deterministic boundary algorithm in Chapter 3 as input.

The three smaller networks are shown in Figure 4.3. Two of them have no useful segmentation. This comes from the medial axis being a ring without vertices. Hence, our scheme provides nothing useful. This is to be expected with such simple



(a) V_2 and V_3 nodes (b) Clusters

Figure 4.4: Clustering the street map example. *Note how the V_2 nodes reflect the street map, and how well the vertex clusters describe the intersections.*

topologies. For the third of the networks, shown in Figure 4.3(b), the clustering works perfectly.

The two large networks feature a complex topology. Consider the V_2 nodes in Figures 4.4(a) and 4.5. In both case, they reflect the medial axis of the explored area very well. The resulting clustering can be seen in Figures 4.4(b) and 4.6. In both cases, it looks as expected, so we conclude that our scheme works well in the scenarios it is intended for.

4.2 Cluster Graphs

While the clusters are built, they get a unique ID. Every node becomes either a member of some $\mathrm{VC}(R) \in \mathscr{R}$ or some $T \in \mathscr{T}$. It stores this cluster's ID, denoted by $\mathrm{id}(v) \in \mathscr{R} \cup \mathscr{T}$, and uses it as location description.

The clusters give rise to the graph $\hat{G}$, which is defined in the following definition.

Definition 4.10. *The* cluster graph *is the undirected graph $\hat{G} = (\hat{V}, \hat{E})$, where*

$$\hat{V} = \mathscr{R} \cup \mathscr{T}, \tag{4.6}$$

$$\hat{E} = \{\mathrm{id}(u)\mathrm{id}(v) : \mathrm{id}(u) \neq \mathrm{id}(v), uv \in E\}\,. \tag{4.7}$$

The network constructs this graph and distributes it among all nodes. By exchanging their cluster IDs with their direct neighbors, a node u can detect that it

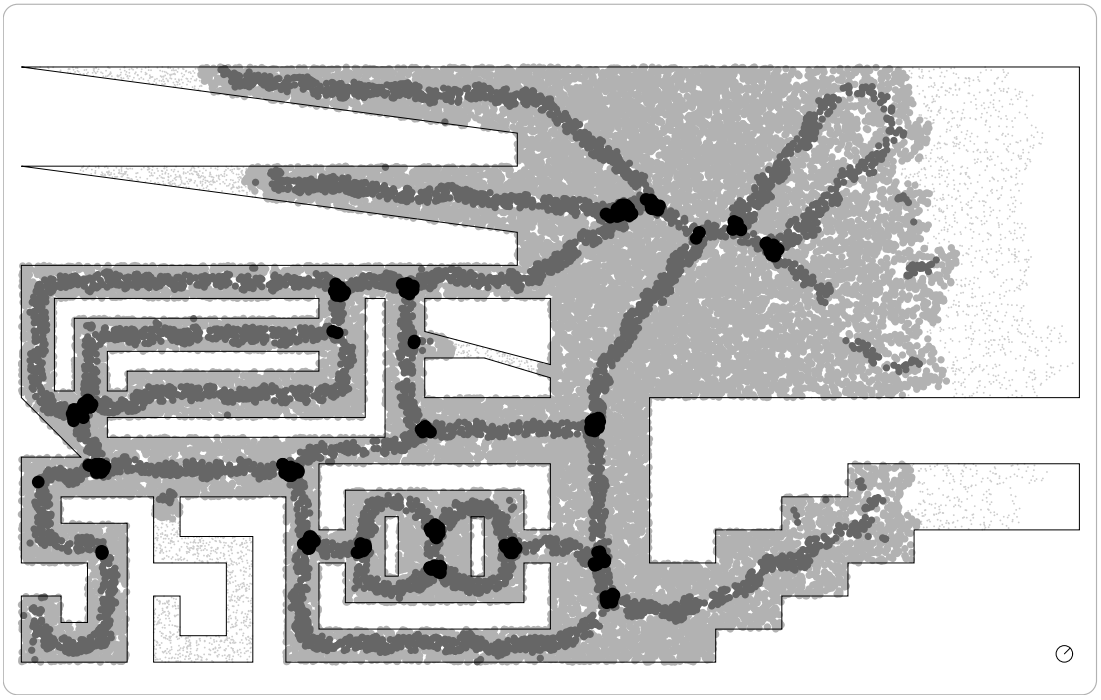


Figure 4.5: V_2 nodes of large network. *The V_2 nodes describe the topology well in the dense areas, but become unstable in the sparse region.*

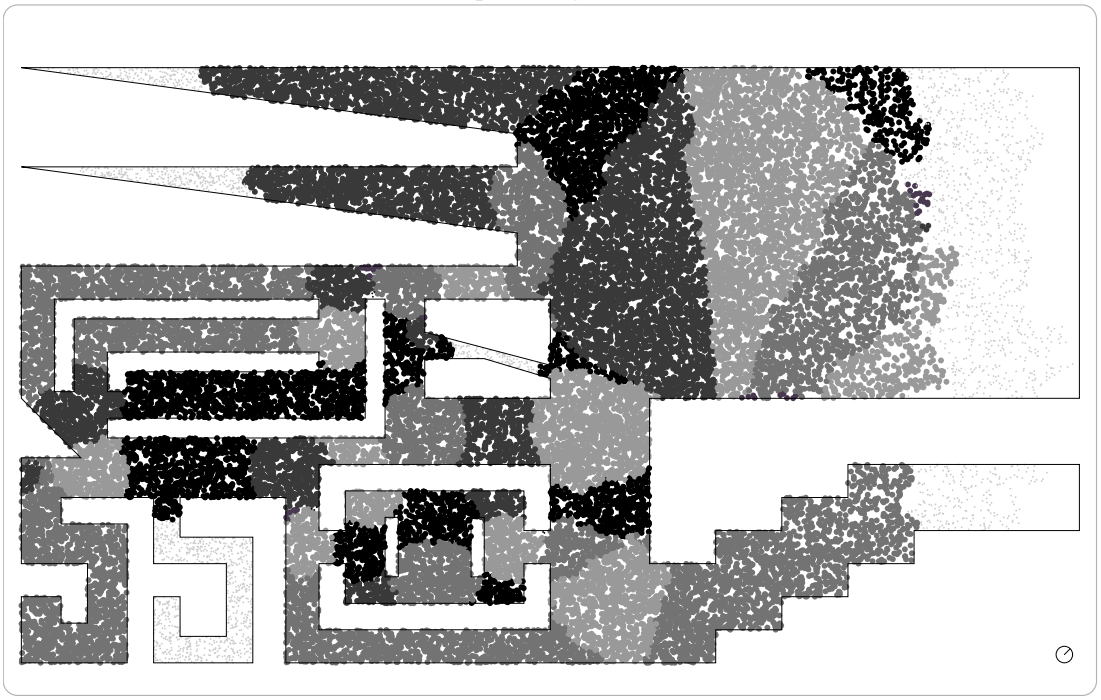


Figure 4.6: Clustering in the large network. *Throughout the explored area, the segmentation has the expected shape.*

has a neighbor v with $\mathrm{id}(u) \neq \mathrm{id}(v)$ and conclude $\mathrm{id}(u)\mathrm{id}(v) \in \hat{E}$. It then broadcasts this cluster edge to all of its neighbors. A node receiving a cluster edge for the first time adds its end nodes to its local copy of $\hat{V}$, if necessary, and stores the edge as well. It then broadcasts the edge.

Eventually, all nodes store a local copy of $\hat{G}$, and they know to which of the graph's vertices they belong. This knowledge becomes our location information.

Weights: The cluster graph $\hat{G}$ provides a simple description of the network topology. It can be useful to further enhance it with weights, based on descriptive properties of clusters.

- By running $(\delta_{\hat{G}}(c))^2/2$ standard network flow algorithms, each node $c \in \hat{V}$ can be turned into the complete graph on its incident edges. For each connection, the maximum throughput and minimum-energy cut can be calculated and stored.

- Similar, minimum and maximum distance between any two exits of a cluster can be determined.

Local Coordinate Systems: While we believe that coordinates should not be used on a global scale, they can be quite useful in the clusters. The clusters have no holes, and the vertex clusters are even convex. These are perfect conditions for any coordinate-based algorithm. Note that we do not require coordinates to have anything to do with the actual embedding, as long as they are useful for the application at hand. One advanced routing scheme that uses similar ideas was already proposed [BGJ05b], using the medial axis as well, based on polar coordinates around medial vertices.

Possibilities for the construction of a location coordinate system (LCS) include the following:

- According to Lemma 4.3, the boundary of a vertex cluster is an alternating sequence of circle chords and circular arcs, all from the same circle. As the length of each segment can easily be determined within the cluster, and the radius of the circle is known, it is trivially possible to reconstruct the exact shape of the cluster. Then, it is easy to construct an Euclidean coordinate system inside the cluster.

- In any cluster c, we can generate $\delta_{\hat{G}}(c)$-dimensional coordinates, where every node stores its hop-distance to each of the neighboring clusters. It simply requires running BFS waves from the nodes at the cluster exits. Such a coordinate systems allows a message, which contains a path in $\hat{G}$ as routing

information, to be efficiently routed in G, because it can take a shortest path in each cluster.

- The previous two schemes do not necessarily allow to find a node in a tunnel cluster, even if its address is known. In this case, two-dimensional coordinates can be employed. The cluster chooses one exit and one boundary line. If there is just one boundary, because the cluster is a dead-end, it splits it in half. The coordinates are then the hop-distance to the exit, and the hop-distance to the boundary piece.

4.3 Application Benefits

A basic application for any clustering is routing: A node s wishes to send one or more data packets to a node t that it not in its direct vicinity. Instead, the packet needs to be relayed over some intermediate nodes. This requires that the network constructs an s-t-path.

It is usually assumed that s knows the position of t in whatever address scheme is in use. Using a Euclidean coordinate system, this may be because t flooded the network with its coordinates, e.g., because it is the base station, has detected some phenomenon, or similar. Even if the target location is known and the localization is consistent, finding a path to t is surprisingly difficult due to the presence of holes between s and t. Many algorithms have been proposed [WW07], but this problem persists.

When the path P is found, it needs to be stored somewhere. This is usually done by letting the nodes on P store their successor, which in turn means that intermediate nodes have to invest memory as long as the communication path between s and t is needed. Furthermore, the path is expressed in terms of individual nodes, necessitating route repair and re-route protocols.

Using our clustering scheme together with an LCS, this is greatly enhanced: If s knows $\hat{G}$ as well as cluster ID and local coordinates of t, it can compute a path in the cluster graph. It then adds the address of t to the message. Routing is now very easy: A node holding the message

1. checks if it is in the same cluster as t. It then routes the packet using a geographic routing scheme on the local coordinates. This is easier than in the general case because clusters are hole-free, and are convex in the local coordinate system.

2. Otherwise, it forwards the message to any neighbor that is strictly closer to the subsequent cluster ID.

This results in a routing scheme with guaranteed delivery, i.e., packets cannot get stuck somewhere. Also, it is immune to failures of individual nodes. Note that messages still can get lost if a node fails while it deals with a packet; this cannot be circumvented.

Chapter 5

Flows

This chapter deals with the Maximum Energy-Constrained Dynamic Flow (ECDF) problem, where a dynamic flow is sought that sends the maximum possible flow in a given time horizon, under additional battery restrictions. Originally, the problem was intended to be used as one of the cluster weights in Section 4.2.

It quickly turned out that the problem is novel, and has significant differences to previously considered problems. It is based on the classic Maximum Dynamic Flow problem, with two new features: First, the limited batteries of the sensor nodes. This makes the problem more difficult, as we prove in Section 5.2. Second, we have uniform transit times, which makes it easier.

Since the problem was not studied before, this chapter includes a complete treatise on the problem, including complexity proofs and a centralized approximation algorithm.

5.1 Problem Definition

We assume that there are two nodes $s, t \in V$ that wish to communicate. We ask for the maximum amount of data that can be sent from s to t in a given time horizon T. We assume that all nodes have a battery from which energy is consumed when communicating. A node whose battery is empty cannot relay data any more, and is then removed from the network.

To give a concise problem definition, we need a little bit of notation. Refer to Section 2.2.4 for an introduction into dynamic flows. An instance of the τ-ECDF problem is of the form $(G, s, t, \tau, T, u, C, c^{\mathrm{s}}, c^{\mathrm{r}})$, where

- $G = (V, E)$ is the underlying network, $s \in V$ is the source, $t \in V$ is the sink, and $s \neq t$.

- $\tau = (\tau_e)_{e \in E}$ defines the transit times, where $\tau_e \in \mathbb{N}_0 \ \forall e \in E$. Data that is sent over an edge uv at time θ is available for further transmission from v at time $\theta + \tau_{uv}$.

- $u = (u_e)_{e \in E}$ denotes the capacities of the undirected edges. Capacities apply to both directions, in the sense that if v sends f_{vw} flow units to w at time θ, w can send up to $u_{vw} - f_{vw}$ back to v in the same round. Data that is sent over an edge does neither interfere with data sent at other times, nor over other edges.

- $C = (C_v)_{v \in V}$ defines the initial node batteries, where $C_v \in \mathbb{R}^+ \cup \{\infty\}$ $\forall v \in V$. We assume $C_s = C_t = \infty$, as this simplifies several arguments and does not exclude any interesting cases.

- $c^{\mathrm{s}} = (c^{\mathrm{s}}_e)_{e \in E}$ and $c^{\mathrm{r}} = (c^{\mathrm{r}}_e)_{e \in E}$ define the energy consumption of sending resp. receiving. When one unit of data is sent from v to w over edge vw, it decreases C_v by c^{s}_{vw} and C_w by c^{r}_{vw}. We assume $c^{\mathrm{s}}_e > 0$ and $c^{\mathrm{r}}_e \geqslant 0$.

An important problem variant is where $\tau_e = 1$ $\forall e \in E$. This appears in wireless communication, where the transit time on an edge is not the time a signal is travelling through the air. Instead, the transits come from the delays in a store-and-forward network, where a node has to receive a packet in one round, then analyze it to determine its next receiver, and forward it in the next round. We call this case the ECDF problem with *uniform transit times*, in short, the 1-ECDF problem.

There are three assumptions that we make:

- Data packets can be split and sent in arbitrary small pieces. Therefore, our problem allows for fractional solutions.

- The nodes have a heavily limited memory. We do not allow storage in the nodes—when a node receives data at time θ, it has to forward it in the following round.

- We ignore the impact of our distributed ECDF algorithm on the nodes batteries. That is, if a node has energy C_v when it starts to solve an ECDF instance, it would have to use $C_v - x$ as battery value in the formulation, where x is the amount of energy it will spend for running the algorithm. To take this into account, we would need an exact model to compute the energy usage by an algorithm; the usual big-O analysis cannot provide this. Inventing such models is a far too complex task for this work, so we simply assume that the nodes have an estimate for C that considers the algorithm's energy cost.

In Section 2.4, we introduced the *geometric* energy function, where the costs for an edge e of geometric length d are $c^{\mathrm{s}}_e = \Theta(d^\alpha)$, and either $c^{\mathrm{r}}_e = \Theta(c^{\mathrm{s}}_e)$ or $c^{\mathrm{r}}_e = 0$. In this chapter, we will frequently use another energy function, where $c^{\mathrm{s}}_e = 1$ and $c^{\mathrm{r}}_e = 0$ $\forall e \in E$. This function is called the *trivial energy model*.

In addition to the problem's data, we use the following notation: We denote by $\mathscr{P}^{st}$ the set of all feasible, simple s-t-paths in G. The length of a path is defined as $\tau(P) := \sum_{e \in P} \tau_e$. With uniform transit times, this equals the number of edges in P. A path is *feasible* if $\tau(P) \leqslant T$. Then, the source can relay data over P in communication rounds 0 to $T - \tau(P)$. $\varrho(P) := T - \tau(P) + 1 \geqslant 1$ denotes the number of times P can be used.

Now let $P = (e_1, e_2, \ldots, e_k)$. We denote by $\tau_e(P)$ the delay after which data travelling P reaches e, i.e.,

$$\tau_{e_i}(P) = \sum_{j=1}^{i-1} \tau_{e_j} \quad \text{for } i = 1, \ldots, k\,. \tag{5.1}$$

For a node $v \in V$, $c^*_{v,P}$ denotes the energy drained from v when one unit of flow is sent over P, i.e.,

$$c^*_{v,P} = \begin{cases} c^{\mathrm{s}}_{e_1} & \text{if } v = s \\ c^{\mathrm{r}}_{e_k} & \text{if } v = t \\ c^{\mathrm{r}}_{e_i} + c^{\mathrm{s}}_{e_{i+1}} & \text{if } \exists i : v \in e_i \wedge v \in e_{i+1} \\ 0 & \text{otherwise} \end{cases}. \tag{5.2}$$

This allows us to give an exact definition of the ECDF problem:

$$\max \sum_{P \in \mathscr{P}^{st}} \sum_{\theta=0}^{T-\tau(P)} x_P(\theta) \tag{5.3}$$

$$\text{s.t.} \sum_{\substack{P \ni e: \\ 0 \leqslant \theta - \tau_e(P) \leqslant T - \tau(P)}} x_P(\theta - \tau_e(P)) \leqslant u_e \quad \forall e \in E, \theta = 0, \ldots, T \tag{5.4}$$

$$\sum_{P \ni v} \sum_{\theta=0}^{T-\tau(P)} c^*_{v,P} x_P(\theta) \leqslant C_v \qquad \forall v \in V \tag{5.5}$$

$$x_P(\theta) \geqslant 0 \qquad \forall P \in \mathscr{P}^{st}, \theta = 0, \ldots, T - \tau(P)\,. \tag{5.6}$$

In this LP, the variable $x_P(\theta)$ describes the amount of flow that starts travelling along P in round θ. The inequalities (5.4) model the edge capacities, and the inequalities (5.5) describe the node battery constraints.

Note that both T and $|\mathscr{P}^{st}|$ can be exponential in the problem's encoding size, and this LP consists of more than $T|E|$ constraints and up to $T|\mathscr{P}^{st}|$ variables. Hence, the LP does provide an exact formulation of the problem, but does not directly lead to efficient algorithms.

Related Work: The ECDF problem is an extension to existing work on dynamic flows; see Section 2.2.4 for an introduction and pointers to relevant literature. The introduction of battery constraints, and the development of distributed algorithms is novel in this field.

In the WSN context, a similar problem was studied: The *Maximum Lifetime Routing* problem. Here, a flow for given demands is sought that maximizes the time until the first node dies. The motivation is that certain nodes collect sensor data and continuously stream them to one or more base stations. All work on this problem models it as following:

> Maximize T such that there exists a static flow for given demands, where the flow consumes no more than a $1/T$ fraction of each battery.

In [ML06, MLL05], flows with one sink are considered. They propose distributed subgradient algorithms on the Lagrangian relaxation, based on the insight that the gradients can be decomposed and computed locally. In [SL04], a multi-commodity flow variant is solved using an exponential penalty function, similar to our approach below. The same problem can also be solved with a combinatorial flow augmentation scheme [CT04]. In another variant [ZS03], a problem with special relay stations between network and data sink is considered. A comparison of several practical protocols can be found in [BHE06].

It should be noted that all of these algorithms provide very little to the ECDF problem. Simply considering static flows that consume just $1/T$ of each battery leaves a large gap to dynamic flows. Consider a network of n nodes, connected in a line, with source and sink at the ends, and $T = n - 1$. Our dynamic flow approach exploits the fact that the source-sink-path can be used exactly once, whereas repeating a static flow would just allocate $1/(n-1)$ of the available battery capacities. So there is a potential gap of size up to $\Theta(n)$.

In our distributed algorithms, we approximate linear programs with an exponential potential function method [Bie02]. This approach was already explored for dominating set LPs [KW05, Kuh05].

5.2 Variant Complexities

In this section, we analyze the complexity structure of different ECDF problem variants. We begin with the case of arbitrary transit times.

Theorem 5.1. *τ-ECDF is* NP*-hard.*

Proof. Consider an instance of the PARTITION problem: Given n positive integers $a_1, \ldots, a_n$ with $\sum_{i=1}^{n} a_i = 2T$ for some $T \in \mathbb{N}$, partition them into two sets of equal

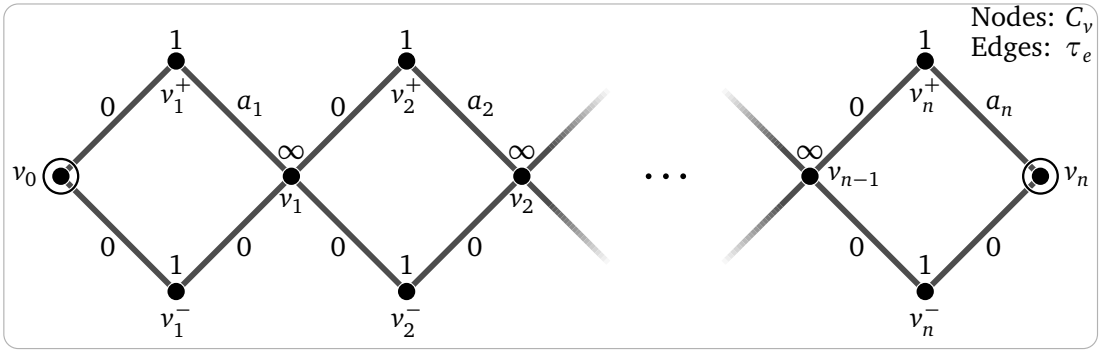


Figure 5.1: Reduction from PARTITION. *Solving this ECDF instance equals finding a partition S under which $\sum_{i\in S} a_i = \sum_{i\notin S} a_i$.*

weight, i.e., find a $S \subset \{1,\dots,n\}$ such that $\sum_{i\in S} a_i = \sum_{i\notin S} a_i = T$. This problem is NP-hard [GJ79].

We claim the PARTITION instance is feasible iff the ECDF instance shown in Figure 5.1 has an optimal solution of value 2, where the horizon is T, the energy function is the trivial one, and all edges have capacity 1.

More than 2 is impossible because each $\{v_i^+, v_i^-\}$ defines a node cut with battery capacity 2. For $S \subset \{1,\dots,n\}$, we define P_S to be the s-t-path that uses v_i^+ wherever $i \in S$ and v_i^- otherwise.

Assume $S \subset \{1,\dots,n\}$ is feasible for the PARTITION instance. Then P_S and $P_{\{1,\dots,n\}\setminus S}$ are two paths of length T each, and they don't use any battery constrained node together. Hence, sending one unit of flow over each of the paths at time 0 defines a feasible ECDF solution.

Next, suppose there is a solution for the ECDF instance that delivers 2 flow units in time. Let $\mathscr{P}$ be the set of flow paths used in the solution, and let x_P denote the total flow sent over path $P \in \mathscr{P}$. Because each $\{v_i^+, v_i^-\}$ defines a saturated node cut, $\mathscr{P} \subseteq \{P_S : S \subset \{1,\dots,n\}, \tau(P_S) \leqslant T\}$. Each v_i^+ is used by some flow paths with total flow 1, so it holds that

$$\sum_{P\in\mathscr{P}} x_P \tau(P) = \sum_{P\in\mathscr{P}} x_P \sum_{i:v_i^+\in P} a_i = \sum_{P\in\mathscr{P}} \sum_{i:v_i^+\in P} x_P a_i = \sum_{i=1}^{n} a_i \sum_{P\in\mathscr{P}:v_i^+\in P} x_P = \sum_{i=1}^{n} a_i = 2T\ .$$

As $\sum_{P\in\mathscr{P}} x_P = 2$ and $\tau(P) \leqslant T$ for all $P \in \mathscr{P}$, this equation can only hold if every $\tau(P)$ equals T. So every $P_S \in \mathscr{P}$ defines a feasible solution S for the PARTITION instance. □

An important property of static network flows is the existence of two popular encoding schemes with polynomial size: First, edge-based, where there is a flow value for every edge. Second, path-based, where there is a flow value for every

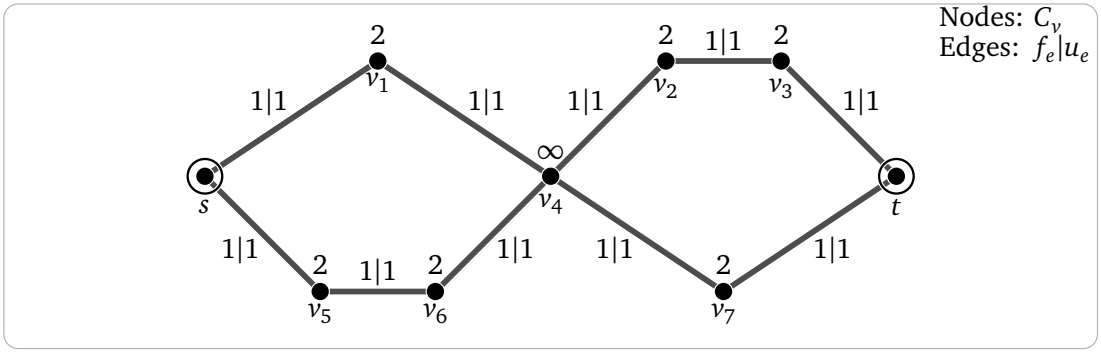


Figure 5.2: Path decompositions do matter. *Different path decompositions lead to different objectives, though they use the same edge flow values.*

path. This even holds for the maximum dynamic flow problem [FF58]. Now we show that both schemes are not applicable to τ-ECDF.

For the edge-based encoding we interpret edge flow values as the maximum flow that is sent over an edge. Consider the network and flow in Figure 5.2. There are four s-t-paths in it: $P_{\cap\cap} = (s, v_1, v_4, v_2, v_3, t)$, $P_{\cap\cup} = (s, v_1, v_4, v_7, t)$, $P_{\cup\cap} = (s, v_5, v_6, v_4, v_2, v_3, t)$ and $P_{\cup\cup} = (s, v_5, v_6, v_4, v_7, t)$. We show that the given solution—with flow value 1 for every edge—has different values depending on the path decomposition. Let $T = 6$. If we use $P_{\cap\cap}$ and $P_{\cup\cup}$, we can use both paths twice, with a total flow of 4. On the other hand, if we use $P_{\cap\cup}$ and $P_{\cup\cap}$, the paths may be used once resp. twice, giving a total flow of 3. The only edge-based solution encoding that we are aware of assigns time-dependent flow functions $f_e : \{0, \dots, T\} \to \mathbb{R}$ to the edges, which are not necessarily of polynomial size.

Unfortunately, even the usual workaround of using a path-based formulation does not help:

Theorem 5.2. *There are τ-ECDF instances that allow no optimal solution consisting of a polynomial number of flow paths.*

Proof. Let $k \in \mathbb{N}$. Consider the ECDF instance $\mathcal{N}_k$ depicted in Figure 5.3. It consists of $k-1$ connected cycles. In the i-th cycle, one can send flow from entry to exit over two paths: Using v_i^1, incurring a delay of 2^{i-1}, or using v_i^0 with zero transit. Both paths can carry a total flow of 2^{k-1}, resulting from battery capacities at the intermediate nodes and the trivial energy consumption model. The edge $v_{k-1}t$ has capacity 1, all other edges have infinite capacity. The horizon is $T = 2^k - 1$. This instance has an encoding size of $O(k^2)$ bits.

Next we construct a solution $x^{(k)} = (x_P(\theta))_{P,\theta}$ for $\mathcal{N}_k$. For $m \in \mathbb{N}_0$, let $b_i(m)$ denote the i-th bit in a binary representation of m, i.e., $m = \sum_{i=0}^{\infty} 2^{b_i(m)}$. Denote by $P_m^k = (s, v_0^{b_0(m)}, v_0, v_1^{b_1(m)}, v_1, v_2^{b_2(m)}, v_2, \dots, v_{k-1}, t)$, $m \in \{0, \dots, 2^k - 1\}$, the path that takes the long-delay route in cycle i if $b_i(m) = 1$ and the zero-transit route

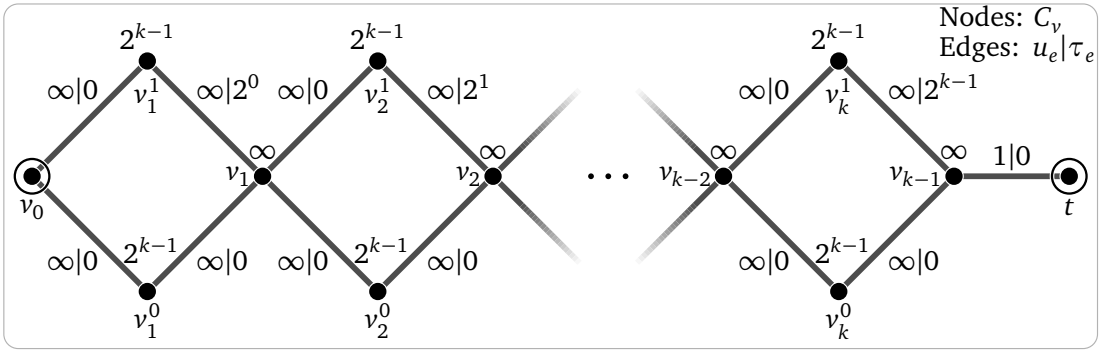


Figure 5.3: No polynomial path decomposition exists. *In the shown network $\mathcal{N}_k$, an optimal solution has to deliver one flow unit to t in every $\theta \in \{0,\dots,T\}$, and must use a different path for each.*

otherwise. In the solution $x^{(k)}$, we send one unit of flow over each of the 2^k paths P_m^k, $m = 0,\dots,2^k-1$ at time 0. Every edge other than $v_{k-1}t$ is used by exactly 2^{k-1} paths, as is every node with limited battery. Because of $\tau(P_m^k) = m$ for all m, the flows pass $v_{k-1}t$ at different times. Hence, $x^{(k)}$ is feasible.

We claim that $x^{(k)}$ is the only optimal solution for $\mathcal{N}_k$, for every $k \geqslant 1$. We prove this by induction over k. For $k = 1$, the claim holds trivially. So assuming it holds for $k-1$, we now prove it for k: Let $y^{(k)} = (y_P(\theta))_{P,\theta}$ be any optimal solution for $\mathcal{N}_k$. The edge $v_{k-1}t$ carries one unit of flow in every time step, as the solution value is $2^k = (T+1)/u_{v_{k-1}t}$.

Because the cut $\{v_{k-1}^1, v_{k-1}^0\}$ is saturated w.r.t. the batteries, exactly half the total flow uses v_{k-1}^1. Because of $\tau_{v_{k-1}^1 v_{k-1}} = 2^{k-1}$, this flow must reach v_{k-2} until time $T - 2^{k-1} = 2^{k-1} - 1$. Furthermore, it can not arrive at v_{k-1} before 2^{k-1}, which leaves these 2^{k-1} flow units exactly 2^{k-1} time steps to pass the unit-capacity edge $v_{k-1}t$. Hence, this edge is saturated from time step 2^{k-1} on. So the other half of the flow, which uses v_{k-1}^0, must pass this edge before 2^{k-1}, which means it must reach v_{k-2} by the time $2^{k-1}-1$. Because the flow gets forwarded to $v_{k-1}t$, which has unit capacity, the flow must reach v_{k-2} as one unit per round.

Hence, each of these halves is a flow that delivers 2^{k-1} flow units to the node v_{k-2}. So we can construct an optimal flow for $\mathcal{N}_{k-1}$ from $y^{(k)}$: Scale the flow by $1/2$, cut the flow paths after node v_{k-2} and add the edge $v_{k-2}t$ in $\mathcal{N}_{k-1}$. By induction this flow must be $x^{(k-1)}$, which sends one unit of flow over each P_m^{k-1}, $k = 0 \dots 2^{k-1}-1$. We can recover $y^{(k)}$ from that: There must be two flow units reaching v_{k-2} at time 0, corresponding to the one unit over P_0^{k-1} in $x^{(k-1)}$. They are divided equally to the two halves, and reach t in time step 0 resp. 2^{k-1}. Now there must be another two units reaching v_{k-2} at time 1, which means they use path P_0^{k-1} or P_1^{k-1}. The former path's total flow in $x^{(k-1)}$ is already assigned, so both units must come over P_1^{k-1}. This can be iterated: The two units reaching v_{k-2} at time θ, $\theta = 2,\dots,2^{k-1}-1$,

must be on path P_θ^{k-1}, because the other possible paths are already exhausted in earlier iteration steps. They are then split into two units that reach t in $\mathcal{N}$ in time steps θ and $\theta + 2^{k-1}$. This proves that in $y^{(k)}$ each path P_m^k is used and carries a flow of 1, hence $y^{(k)} = x^{(k)}$. □

Now that we established that τ-ECDF is hard, and gave a reasoning why we do not even see a polynomial solution encoding scheme, we turn to the more interesting special case, the 1-ECDF problem. Unfortunately, we do not know whether this problem is also hard. All that we know is that it admits an FPTAS (see proof in Section 5.3), and that the integer variant is hard:

Theorem 5.3. *Finding a 1-ECDF solution with integral flow values is* NP*-hard.*

Proof. By reduction from the following 3-PARTITION variant: Given three sets of positive integers $\{a_1, \ldots, a_n\}$, $\{b_1, \ldots, b_n\}$, and $\{c_1, \ldots, c_n\}$ with $\sum_{i=1}^n a_i = \sum_{i=1}^n b_i = \sum_{i=1}^n c_i =: L$, find a partition into n triples of equal weight, i.e., find permutations $\alpha, \beta, \gamma \in S_n$ such that $a_{\alpha(i)} + b_{\beta(i)} + c_{\gamma(i)} = 3L/n$ for all i. This problem is strongly NP-hard [GJ79].

We construct an ECDF instance $G = (V, E)$ as follows: For each of the $3n$ numbers, say a_i, there is a chain A_i consisting of a_i nodes connected in line, of which a_i^+ is the first and a_i^- is the last. Additionally, there are the source s and sink t. The source is connected to each entry node of the first set, that is, $sa_i^+ \in E$ for all i. Each exit node of the first set is connected to each entry of the second: $a_i^- b_j^+ \in E$ for all i, j. Analogously, the exits from the second set are linked to the entries of the third, and all exits from the third are linked to t. Each edge $e \in E$ has unit capacity $u_e = 1$. Furthermore, each node $v \neq s, t$ has a battery capacity of $C_v = 1$, where the power consumption model is the trivial one: $c_e^{\mathrm{s}} = 1, c_e^{\mathrm{r}} = 0$ for all $e \in E$. The time horizon is $T = 3L/n + 1$.

We claim that the 3-PARTITION instance is solvable iff the optimal integral ECDF solution value is n. There cannot be a total flow of more than n because $\{a_1^+, \ldots, a_n^+\}$ forms a node cut with total energy n.

First assume that (α, β, γ) is a feasible solution for the 3-PARTITION instance. Then there are n paths $P_i = (s, A_{\alpha(i)}, B_{\beta(i)}, C_{\gamma(i)}, t)$, $i = 1, \ldots, n$. These paths are pairwise node-disjoint except for s and t, and each has length $\tau(P_i) = a_{\alpha(i)} + b_{\beta(i)} + c_{\gamma(i)} + 1 = T$. So we can send one unit of flow at time 0 over each of these paths, resulting in a total flow of n.

For the other direction assume there is a feasible and integral ECDF solution $(x_P(\theta))_{P \in \mathscr{P}, \theta \in \theta(P)}$, where $\mathscr{P} \subseteq \mathscr{P}^{st}$, $\theta(P) \subseteq \{0, \ldots, T - \tau(P)\}$, and $x_P(\theta) \in \mathbb{N}^+$ for all $P \in \mathscr{P}, \theta \in \theta(P)$. Assume it has value n. Because each of the sets $\{a_1^+, \ldots, a_n^+\}$, $\{a_1^-, \ldots, a_n^-\}$, $\{b_1^+, \ldots, b_n^+\}$, $\ldots$, $\{c_1^-, \ldots, c_n^-\}$ is a saturated node cut, it cannot be crossed twice by any path.

Hence, each $P \in \mathscr{P}$ is one of the paths $P_{i,j,k} = (s, A_i, B_j, C_k, t)$, $i, j, k = 1, \ldots, n$. Since the flow is integral and all battery capacities are 1, each $x_P(\theta) = 1$. Each of the chains in the graph can be used by just one flow path due to its battery capacity, and because the total flow is n, each chain is used by exactly one path. So $\mathscr{P} = \{P_{\alpha(i),\beta(i),\gamma(i)} : i = 1, \ldots, n\}$ for some $\alpha, \beta, \gamma \in S_n$. We know that

$$\begin{aligned}
\sum_{i=1}^{n} \tau(P_{\alpha(i),\beta(i),\gamma(i)}) &= \sum_{i=1}^{n} (a_{\alpha(i)} + b_{\beta(i)} + c_{\gamma(i)} + 1) && (5.7)\\
&= \sum_{i=1}^{n} a_i + \sum_{i=1}^{n} b_i + \sum_{i=1}^{n} c_i + n && (5.8)\\
&= 3L + n && (5.9)\\
&= nT\,, && (5.10)
\end{aligned}$$

and, because no path length can exceed T due to the feasibility of the solution, we conclude that each path length must equal $T = 3L/n + 1$. Therefore $a_{\alpha(i)} + b_{\beta(i)} + c_{\gamma(i)} = 3L/n$ for each $i = 1, \ldots, n$, proving that α, β, γ is feasible for the 3-PARTITION instance. □

Unfortunately, the proof does not carry over to the fractional ECDF problem: There is a trivial LP formulation for the ECDF instance that uses the n^3 possible flow paths explicitly.

5.3 Centralized Approximation

In this section, we concentrate on 1-ECDF and show that it does admit an FPTAS. For that matter, we distinguish different cases based on the horizon T. The first case is easy to solve:

Lemma 5.4. *1-ECDF can be solved in polynomial time, if T is polynomially bounded in n.*

Proof. The time-expanded graph $G(T)$ (see Section 2.2.4) has polynomial size and therefore allows a simple edge-based LP. □

A temporally repeated flow is a flow $(x_P(\theta))_{P,\theta}$ where there is no variation over time in a path's flow amount, i.e., $x_P(\theta) = x_P(\theta')$ for all $\theta, \theta' \in \{0, \ldots, T - \tau(P)\}$. When $T > 2n$, the problem of finding a maximum temporally repeated 1-ECDF

solution can be formulated as follows; recall that $\varrho(P) = T - \tau(P) + 1$:

$$\max \sum_{P \in \mathscr{P}^{st}} \varrho(P) x_P \tag{5.11}$$

$$\text{s.t.} \sum_{P \ni e} x_P \leqslant u_e \qquad \forall e \in E \tag{5.12}$$

$$\sum_{P \ni v} \varrho(P) c^*_{v,P} x_P \leqslant C_v \quad \forall v \in V \tag{5.13}$$

$$x_P \geqslant 0 \qquad \forall P \in \mathscr{P}^{st}\,. \tag{5.14}$$

The restriction $T > 2n$ comes from inequality (5.12), which is only valid if all the paths that use some edge e send their flow over e in at least one common point in time.

The size of this LP is not polynomially bounded. Fortunately, it can be solved using a separation approach:

Lemma 5.5. *Maximum temporally repeated solutions for 1-ECDF with $T > 2n$ can be found in polynomial time.*

Proof. The dual LP of (5.11)–(5.14) is

$$\min \sum_{v \in V} C_v \mu_v \qquad + \sum_{e \in E} u_e \pi_e \tag{5.15}$$

$$\text{s.t.} \sum_{v \in P} \varrho(P) c^*_{v,P} \mu_v + \sum_{e \in P} \pi_e \qquad \geqslant \varrho(P) \;\; \forall P \in \mathscr{P} \tag{5.16}$$

$$\mu_v \geqslant 0 \;\forall v, \qquad \pi_e \geqslant 0 \;\forall e\,. \tag{5.17}$$

The separation problem for this LP is to find a violated inequality (5.16), given edge weights $(\pi_e)_{e \in E}$ and node weights $(\mu_v)_{v \in V}$: Find a path $P \in \mathscr{P}^{st}$ satisfying

$$\sum_{v \in P} c^*_{v,P} \mu_v + \sum_{e \in P} \frac{1}{\varrho(P)} \pi_e < 1 \tag{5.18}$$

or prove that no such path exists. The left-hand-side of (5.18) can be rewritten as

$$\sum_{v \in P} c^*_{v,P} \mu_v + \sum_{e \in P} \frac{1}{\varrho(P)} \pi_e = \sum_{uv \in P} \left(\tfrac{1}{T - \tau(P) + 1} \pi_{uv} + c^s_{uv} \mu_u + c^r_{uv} \mu_v \right), \tag{5.19}$$

which is just the length of P according to some length-dependent edge weights. So the separation problem reduces to the question whether the shortest path in $\mathscr{P}^{st}$ according to this weight function has a length strictly less than 1.

Because $\tau(P) \in \{1, \ldots, n\}$ for each $P \in \mathscr{P}^{st}$, there are just n possible values for $1/(T - \tau(P) + 1)$. We can find the shortest path by enumerating over these values. In each step, we seek the shortest path consisting of exactly k edges for some $k \in \{1, \ldots, n\}$. This can be done in polynomial time by searching for the shortest path from $s(0)$ to $t(k)$ in the time-expanded graph $G(k)$. □

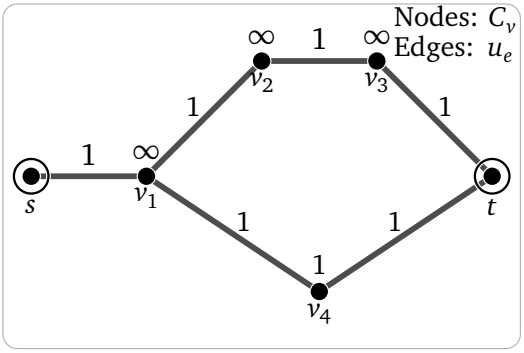


Figure 5.4: Gap using temporally repeated solutions. *In this network, optimal solutions are never temporally repeated.*

While we do intend to use optimal temporally repeated solutions to find good solutions for 1-ECDF, we do have to live with a gap:

Lemma 5.6. *Temporally repeated 1-ECDF solutions are not always optimal.*

Proof. Consider the network shown in Figure 5.4. The horizon is $T = 4$, communication cost is the trivial one. There are two paths in this network: The "upper" one $P = (s, v_1, v_2, v_3, t)$ that can be used exactly once, and the "lower" one $Q = (s, v_1, v_4, t)$, that can be used twice with a total flow of 1 due to the battery limitation at v_4. An optimal solution sends one flow unit along P at time 0 and another unit along Q at time 1, with a total flow value of 2. Optimality holds because edge sv_1 is saturated at all times.

A temporally repeated solution sends x_P along P at time 0 and x_Q along Q at times 0 and 1. Because of the capacity of sv_1, $x_P + x_Q \leqslant 1$ holds. Furthermore, $2x_Q \leqslant 1$ due to the battery capacity at v_4. The total flow is $x_P + 2x_Q$, which is maximized by $x_P = x_Q = 1/2$ with an objective value of $3/2$. □

So far, we know that we can construct solutions of a certain type, and we know they are not always optimal. But fortunately, we can bound the optimality gap:

Lemma 5.7. *For $T > \lambda n$, $\lambda \geqslant 2$ the value of a maximum temporally repeated solution* TR *is greater than or equal to $(\lambda - 1)/\lambda$* OPT*, where* OPT *be the value of an optimal 1-ECDF solution.*

Proof. Let $x = (x_P(\theta))_{P,\theta}$ be an optimal solution. We construct a temporally repeated solution $y = (y_P)_P$ from it by averaging over all path flows. So let

$$y_P := \tfrac{1}{\varrho(P)} \sum_{\theta=0}^{T-\tau(P)} x_P(\theta) \tag{5.20}$$

for each $P \in \mathscr{P}^{st}$. This flow satisfies all battery capacity constraints, because each flow path carries the same total flow as in x, and it delivers the same flow within the horizon. It may violate edge capacities though. So let $e \in E$. Then the load on e at time θ is

$$\begin{aligned}
\sum_{\substack{P \ni e: \\ 0 \leqslant \theta - \tau_e(P)}} y_P &\leqslant \sum_{P \ni e} y_P && (5.21)\\
&= \sum_{P \ni e} \frac{1}{\varrho(P)} \sum_{\theta=0}^{T-\tau(P)} x_P(\theta) && (5.22)\\
&\leqslant \frac{1}{T-n} \sum_{P \ni e} \sum_{\theta=0}^{T-\tau(P)} x_P(\theta) && (5.23)\\
&\leqslant \frac{1}{T-n} T u_e && (5.24)\\
&\leqslant \frac{\lambda}{\lambda-1} u_e \,. && (5.25)
\end{aligned}$$

Hence, $\frac{\lambda-1}{\lambda} y$ is a feasible temporally repeated flow. □

Putting things together, we have a gap that is shrinking when T becomes large, and an optimal algorithm for small T. So all that is left to do is to decide where to switch between the two algorithms. This leads to the main theorem of this section:

Theorem 5.8. *1-ECDF admits an FPTAS.*

Proof. Let $\varepsilon > 0$. If $T \leqslant n/\varepsilon$ or $T \leqslant 2n$, we can solve the problem directly by Lemma 5.4. Otherwise, by Lemma 5.5 we can compute a maximum temporally repeated flow in polynomial time and, by Lemma 5.7, its value is at least

$$((\tfrac{1}{\varepsilon} - 1)/\tfrac{1}{\varepsilon})\mathrm{OPT} = (1-\varepsilon)\mathrm{OPT}\,. \tag{5.26}$$

□

5.4 Distributed Approximation

In this section, we propose a distributed FPTAS for 1-ECDF. The core idea is to find a clever way to implement the centralized algorithms from the previous section in a distributed fashion. The biggest obstacles are obviously that we rely on linear programming, and especially the ellipsoid method, as we solve LPs by separation. In Section 2.2.3, we introduced the LP approximation algorithm by Garg

and Könemann [GK98]. This algorithm approximates fractional packing LPs, using a separation oracle. Furthermore, the algorithm allows for a simple distributed implementation.

Similar to the previous section, we solve 1-ECDF by distinguishing two cases: $T > n/\varepsilon$, where the optimality gap of temporally repeated solutions is small, and $T \leqslant n/\varepsilon$, where the horizon is polynomially bounded. While the Garg and Könemann algorithm can easily be distributed, it requires a fast approximation for the dual separation problem. For this purpose, we show how to reduce the n shortest path computations we needed in the proof of Lemma 5.5 to one:

Lemma 5.9. *Let $T > \lambda n$, $\lambda \geqslant 2$. Let $\pi_e \geqslant 0$ for all $e \in E$ and $\mu_v \geqslant 0$ for all $v \in V$. Then the dual separation problem* (5.18) *for temporally repeated flows can be $\lambda/(\lambda-1)$-approximated using a single shortest path computation.*

Proof. The separation problem is to find a shortest path according to the length function

$$z(P) := \sum_{uv \in P} \left(\tfrac{1}{\varrho(P)}\pi_{uv} + c^s_{uv}\mu_u + c^r_{uv}\mu_v\right). \tag{5.27}$$

We define another function to approximate z:

$$y(P) := \sum_{uv \in P} \left(\tfrac{1}{T}\pi_{uv} + \tfrac{\lambda-1}{\lambda}(c^s_{uv}\mu_u + c^r_{uv}\mu_v)\right). \tag{5.28}$$

Observe that for each $P \in \mathscr{P}^{st}$,

$$T \geqslant \varrho(P) \geqslant T - n > T - \tfrac{1}{\lambda}T = \tfrac{\lambda-1}{\lambda}T \tag{5.29}$$

$$\implies \quad \tfrac{1}{T} \leqslant \tfrac{1}{\varrho(P)} \leqslant \tfrac{\lambda}{\lambda-1}\tfrac{1}{T}. \tag{5.30}$$

This proves that $y(P) \leqslant z(P) \leqslant \lambda y(P)/(\lambda-1)$ for every $P \in \mathscr{P}^{st}$. Therefore, the minimum-y path is a $\lambda/(\lambda-1)$-approximation to the minimum-z path. Now $y(P)$ is just a sum of directed, non-negative edge weights. Then, the minimum-y path can be found by a single run of any shortest path algorithm. □

Turning the algorithm by Garg and Könemann into a distributed algorithm for the large-T case is now straightforward, see Algorithm 5.1:

Lemma 5.10. *Let $\varepsilon > 0$ and $T > n/\varepsilon$, $1/\varepsilon \geqslant 2$. Then Algorithm 5.1 is a $(1-\varepsilon)^{-4}$-approximation for the ECDF problem and runs in time $O(n(n+m)\frac{1}{\varepsilon}\log_{1+\varepsilon}(n+m))$.*

Proof. According to Lemma 5.9, the Bellman-Ford algorithm computes an approximation to the dual separation problem with ratio $\lambda/(\lambda-1) = (1-\varepsilon)^{-1}$. Hence, Algorithm 5.1 is a $(1-\varepsilon)^{-3}$-approximation to finding a temporally repeated ECDF solution (Theorem 2.1). Because a maximum temporally repeated flow is a $(1-\varepsilon)^{-1}$-approximation to the original ECDF problem (Lemma 5.7), the solution found by

Algorithm 5.1: Distributed algorithm for $T > \frac{1}{\varepsilon}n$

1 Each node $v \in V$ initializes and stores μ_v and π_e for each $e \in \delta(v)$, according to Algorithm 2.1
2 **repeat**
3 s initiates a distributed Bellman-Ford shortest path algorithm following Lemma 5.9
4 The network reports the approximate shortest path's weight and capacity to s
5 The network augments flow along this path, each nodes updates the dual weight, and reports the dual objective increase back to s
6 **until** *s observes that the dual objective is at least* 1
7 Scale flow for feasibility, unless already done during augmentation

Algorithm 5.1 is a $(1-\varepsilon)^{-4}$-approximation. The runtime results from the iteration bound of Theorem 2.1 and the $O(n)$ runtime of a distributed Bellman-Ford computation. □

The other case, with small T, is mostly analogous, so we do not repeat everything in detail:

Lemma 5.11. *Let $\varepsilon > 0$ and $T \leqslant \frac{1}{\varepsilon}n$. Then there is a distributed algorithm that finds a $(1-\varepsilon)^{-2}$-approximation in time $O(\frac{1}{\varepsilon^2}mn^2 \log_{1+\varepsilon}(\frac{1}{\varepsilon}mn))$.*

Proof. Run a variant of Algorithm 5.1 on the exact LP formulation (5.3)-(5.6). □

Together, Lemmas 5.10 and 5.11 (and the special case $\varepsilon \geqslant \frac{1}{2}$, $T > \frac{1}{\varepsilon}$, which is trivial to resolve) allow us to state the main theorem of this section:

Theorem 5.12. *ECDF admits a distributed FPTAS. Each node $v \in V$ needs to store no more than $O(p(v)+1)$ many variables, where $p(v)$ denotes the number of flow paths using v in the solution.*

Proof. The only point that is left to prove is the storage. Notice that even in the case with $T \leqslant n/\varepsilon$, the nodes do not have to store all dual weights, but only those that have been changed from their initial value. As each such change coincidences with a flow path being routed through a node, the claimed bound holds. □

5.5 Problem Extensions

Finally, we discuss two important variants of the problem.

Multi-terminal variants: ECDF problems where there is one source and many sinks can be solved by our algorithms, both centralized and distributed, as well. It is sufficient to alter $\mathscr{P}^{st}$ accordingly. The opposite case with one sink and many sources can be solved by exchanging them and reversing time. This just applies to the objective of maximizing the total flow though, as max-min objective variants are no longer fractional packing problems. The situation is similar for multi-commodity settings with many sources and sinks: Maximizing the total flow is possible by adjusting $\mathscr{P}^{st}$—in the distributed setting an additional syncing step between the sources is needed in each iteration. The max-min multi-commodity case can not be solved using our algorithms.

Geometric communication cost functions: In wireless networks, it is a common assumption that the sending cost for transmitting over a link e of length d is $c_e^s = \Theta(d^\alpha)$ for some constant $\alpha \in [2,6]$. The cost for receiving is often modelled as either 0 or $\Theta(c_e^s)$. Our constructive results work for any cost function. We have stated the negative results in Section 5.2 using the trivial cost function for clarity. Note that the problem instances used in the proofs can be embedded such that every link has length 1 (actually, the figures show such embeddings), where the geometric cost function becomes the trivial one. Hence, these results apply as well.

Chapter 6

Shawn

The algorithms for boundary detection and clustering in Chapters 3 and 4 were analyzed using simulations. We used a simulation software that we developed during the course of the research project SwarmNet[1]. This chapter describes the sensor network simulator Shawn[2] [KPB+05, FKFP07]. Shawn is open source software, under the BSD license, and available from SourceForge[3].

There exists a great variety of WSN simulators, refer to the Simulation Tool Comparison Matrix provided by the CRUISE project[4] for details and additional pointers. We will not engage in attempts to run comparative performance tests with other tools, as Shawn is fundamentally different in many aspects, as described in Section 6.1. This limits the usefulness such comparison. There is a poster [FKFP07] and a dissertation [Pfi07] containing comparative charts that show that Shawn is a thousand times faster than two prominent simulators, Ns-2 and TOSSIM. In the description of Shawn's details, we will point out where the underlying design is fundamentally different between simulators, and why this allows Shawn to be so fast.

The focus on speed comes from the applications we had in mind. The topological algorithms we present in this thesis are only applicable on very large networks. We evaluated software that already existed, only to find out that none of them could handle networks larger than a mere 1000 nodes within reasonable time. A simulation where a thousand nodes send a few hundred packets to their immediate neighbors took a full day to run in Ns-2 [FKFP07]. Hence, simulating our networks required building a new tool from scratch, where extreme adaptability and speed are the main focus. Shawn can easily handle up to 500 000 nodes. Most of the simulations we ran for this thesis, on networks with up to 50 000 nodes, finished within five minutes on a standard desktop PC.

[1] `http://www.swarmnet.de`, supported by DFG Focus Program 1126, "Algorithmic Aspects of Large and Complex Networks", Grants Fe 407/9-2 and Fi 605/8-2.

[2] "Shawn" is a name, not an acronym.

[3] `http://shawn.sourceforge.net`

[4] `http://www.ist-cruise.eu`

6.1 Design Principles

Shawn was built around three basic design principles, which are described now. They turned out to be the major reason for Shawn's scalability and adaptability.

Simulate the Effects: Most simulation frameworks attempt close resemblance of "the reality" by mimicking all aspects of it at a very fine-grained level.

For example, sending a message between nodes usually means that it has to be encoded as datagrams, which are then passed through all layers of a complete OSI stack, with simulated variants of all protocols. It is then sent over a simulated "wireless channel", which involves detailed computations of signal propagation, interference effects, and retransmissions of lost signals. On the application level however, the result of all these computations is simply whether and when the message is available at the receiving node. The same result can be obtained by a simple stochastic process that drops or delays messages, given a carefully chosen random model that takes locality—and possibly interference—into account.

Besides the huge computational effort, another drawback is that the approach of modeling all real-world factors can only simulate current technology. The only way for the validation of such model is to compare them with a WSN that exists today. This stands in contrast to the research focus on future networks with hard- and software that will exist in 5 to 10 years from now. The hope behind such simulation models is, of course, the hope that future developments will behave similar to today's. Unfortunately, algorithm developers for sensor networks are not the only researchers in the field. At the same time, researchers in communications, hardware, low-level communication protocols, and so on, do their very best to ensure that future technology will behave much different than today's.

This lead to Shawn's design principle "*Simulate the Effects*": There are interfaces that describe effects on a very abstract level and reference implementations, simulating them as simple as possible. In case somebody needs a simulation of a complete OSI stack with his favorite protocols and physical models, it is easy to integrate that into his simulations. But fortunately there is no need to use such a computationally expensive model.

Centralized vs. Distributed Implementations: Another important principle underlying Shawn is that there is no distinction between centralized and distributed algorithms. Even when developing a distributed algorithm, it is often useful to first implement it as a centralized algorithm, then turn it step-by-step into a distributed protocol. For example, consider a distributed protocol that communicates over a spanning tree. The most convenient way to simulate such an algorithm would be to first construct the tree using a traditional, centralized algorithm, and then let

the distributed protocol implementation use it. There is no benefit in adding a distributed tree algorithm. Actually, doing so would harm the development process as it slows down the simulation.

Therefore, Shawn offers a traditional, graph-like interface to centralized algorithms, and another interface for distributed protocols. An implementation can switch between centralized and distributed mode at any time. Since the full network with protocol implementations and the pool of centralized tasks is accessible, information interchange between both modes is fast and easily implemented. Another interface for such tasks is the tag system (see Section 6.2.3), which is slightly slower as a matter of principle.

Optional Components: The third principle behind Shawn is the belief that it should be possible to turn off any part of the simulation to gain speed. Shawn aims at sensor network applications, as opposed to low-level protocols. A simulation of an application is usually run to see whether an algorithm actually works, and what its output is. Frequently, this can be determined even if communication and routing are completely turned off, and instead messages just appear at their destination in an instant. So the user can change the routing model to a trivial one, allowing him to get answers quickly. Later, when runtime behavior and reliability become the focus of interest, he can switch back to a more complete simulation.

6.2 Simulator Details

Shawn is mostly written in ANSI-C++. It is portable and contains no dependency on particular operating systems or architectures. Therefore, it runs under Windows, Unix/Linux, and Mac OS. Porting it to other systems that provide a decent C++ compiler should pose no problem. The choice of a BSD-style license makes it possible to use Shawn in commercial projects; this separates Shawn from other tools, which usually restrict free use to academia.

The simulator consists of three distinct parts:

- The core library is self-contained and provides all basic tools and models for a simulation, including network models, communication models, a pool for user-provided configuration values, and I/O.
- There are extension modules that can be included in the build. They add features like visualizations, environment topologies, or sensor readings. See Section 6.2.3 for a description of some of them.
- The user front-end is completely separated from the above. There are two alternative front-ends. There is a simple configuration file language that allows

to define variables and spawn simulation tasks. Another option is a Java-inspired scripting language that allows for more complex simulation setups.

6.2.1 Central Components

The two most important parts of the core library are the representation of the network and the event scheduler. They are briefly described now.

Network Representation: The container for a network is a *World* instance. It contains all nodes and provides access to the active models of the simulation. Each node has a position and an arbitrary number of *message processors*. In order to support mobility, the position is a function object that can be queried for current location, movement direction, and speed. In static networks it is simply a constant function.

A message processor is responsible for a certain protocol. An implementation of an algorithm or protocol usually consists of one processor class. Every node stores an instance of this class. Whenever a node receives a message, it presents it to all of its processors until it finds one that accepts the message. This way, multiple protocols can run on the same network without the need to alter code. An application protocol can transparently run with either a complex routing protocol or immediate message delivery.

Time in the Simulation: A distributed simulation needs a notion of time. Shawn contains a discrete *event scheduler*, based on a priority queue. There is no fixed unit associated with time. Events can happen at any time at the precision of the machine's floating point numbers. In distributed mode, objects of the simulation can queue events in the scheduler. The scheduler activates them one by one, thereby advancing the global clock to one event time after the other. This means that idle nodes do not consume simulation time, which is a speed-up against simulators where time is modeled in fixed steps, usually at 10ms intervals, and a callback gets invoked for every node in every step.

In addition to this fine-precision events, there is a periodic event. In it, a callback is invoked for every node and the world object. Note that this is only a slowdown when it is actually needed, because there is no fixed interpretation of time units, i.e., how often this periodic event happens. This system is however necessary because many sensor network application deal with "continuously" sensing the environment.

The choice of a arbitrary-precision time has large impact on many aspects of Shawn. For example, mobility is implemented by letting a node's position be a function object, so that a node can actually move smoothly. We believe this is a

great improvement compared to simulators where a moving node has to jump from one discrete point to another.

6.2.2 Communication- and Graph-Related Models

There is no direct representation of the communication graph. Instead, we use two separate classes, one that describes the basic adjacency relation, and another one that provides neighborhoods via iterators. Together, they define the communication graph. We see an edge in this graph just as a necessary condition for communication. Whether, and when, a particular message arrives at its destination is controlled by a third class.

Communication Models: The *communication model* is essentially a function $V \times V \to \{0,1\}$ that defines the edges of the communication graph, that is, whether communication between two given nodes is possible. It is not required to be a constant function, the existence of edges may change over time, e.g., when nodes move. A communication model may provide an upper bound to the length of edges, i.e., a maximal communication range. This speeds up geometric data structures, most notably the grid-based edge models below.

Besides the abstract base class for communication models, Shawn's core library provides a number of ready-to-use standard communication models. These are:

- In the *Disk Graph Model*, there is a uniform communication range r. Communication between any two nodes is possible whenever their Euclidean distance is at most r.
- The *Permalink Model* is a generic graph-like model. All edges in the graph are defined by the user and do not change.
- The *Radio Irregularity Model* (RIM) is a stochastic model that resembles certain real-world characteristics closely [ZHKS04]. It is actually a special case of QUDGs: there is a direction-dependent range function $r : [0, 2\pi] \to [p, q]$, so the graph is a $\frac{p}{q}$-QUDG. The function r is the output of a particular random process.

Another interesting model is a meta-communication model that wraps multiple user-defined communication models, making it possible to use different models for nodes of different characteristics in the same network.

Edge Models: The previously described communication model can be seen as an oracle that answers the question "$uv \in E$?" for any two nodes $u, v \in V$. The *edge*

model uses this oracle in order to build a data structure that can be queried for neighborhoods of nodes. It does so by providing iterators over $N(v)$ for each $v \in V$. There are multiple edge models in Shawn's core, they differ in speed, memory usage and support for mobility.

- The most basic is the *Simple Edge Model*. It's neighborhood iterator simply queries the communication model n times, once for every potential communication partner. Hence, it has a constant-size memory footprint and needs constant time for initialization. Another benefit is that it works for arbitrary mobility, and dynamic joins and leaves of nodes. This comes at the cost of $O(nC)$ time to query for a neighborhood, where C is the complexity of the communication model.

- The second model is the *List Model*. At network construction time, it evaluates all edges by querying the communication model for every pair of nodes, and stores the resulting graph in adjacency lists. Hence, it is not suited for dynamic networks (although this could be partially supported with $O(\Delta)$-time operations to add or remove nodes). This model needs $O(n^2C)$ time to construct its data structures and stores $O(n\Delta)$ list items. The obvious benefit is the query time, which is the optimal $O(|N(v)|)$ to query $N(v)$.

- The *Grid Model* utilizes a geometric data structure. It partitions the plane into a dynamically growing number of cells. For each cell, the model maintains a list of all nodes that are currently located in it. When queried for a neighborhood, the model uses the communication range bound provided by the communication model to decide which cells may actually contain neighbors. It then iterates over the nodes in these cells, skipping nodes that are not reachable according to the communication model. In the worst case—when all nodes are very close together, falling into the same cell—the grid model behaves just like the simple model. In practice it is much faster, as almost all realistic networks cover some space. The grid model features full support for mobility: There is a callback mechanism by which nodes know the dimensions of the cell they are in, so that they can be moved between cell lists whenever necessary.

- The *Fast List Model* attempts to combine the strengths of the list and the grid model. It is essentially a list model variant. To speed up the initialization, it utilizes a temporary grid model to which all nodes are initially passed. When the network is complete, the model queries the grid model once for all edges and stores the neighborhood in adjacency lists.

The user is free to choose whatever edge model he prefers. Not surprising, practical experience shows that the fast list model is preferable if the network is static and

one can afford the storage space, otherwise the grid model should be used.

Transmission Models: While the previous two models deal primarily with the communication graph, the *Transmission Model* focusses on individual messages. When a node sends a message, it gets passed to the transmission model, which in turn may process it in any way it prefers. There is a built-in distinction between unicast (to a direct neighbor) and broadcast messages (to the complete neighborhood).

- The *Reliable Transmission Model* delivers all messages, either immediately or at the beginning of the subsequent communication round.
- The *Random-Drop Transmission Model* drops messages uniformly with a user-defined probability.
- The transmission model is the obvious place to implement MAC layer effects and protocols, i.e., to simulate the outcome of practical message transmission schemes running under physical influences. Shawn contains ready-to-use implementations of some standard protocols: Aloha (and Slotted Aloha), CSMA (and Zigbee-CSMA), and MACA.

Furthermore, there is mechanism to group transmission models together. Then a message only gets delivered when all chained models agree. This allows for a simple user-configurable way to simulate parts of the OSI stack, by providing one transmission model for every layer. It also allows heterogeneous networks, where each transceiver class is simulated by a different transmission model.

6.2.3 Shawn Extensions

In addition to the core library, Shawn provides several noteworthy features for different applications. Some of them are summarized here.

CGAL Interface: Shawn can be configured such that its internal vector representation is auto-convertible to CGAL's types. CGAL[5] is a C++ library dealing with computational geometry. It provides many geometric data structures and algorithms, which are thus easily integrated into Shawn.

Persistence: There is a convenient way to add persistent data to the network. Both the world and the nodes are containers for so-called *tags*. A tag is a typed pair of name and value. Additionally, it can be nested. Shawn automatically saves

[5] `http://www.cgal.org`

and loads tags together with its network files. This way, it is easy to implement algorithms that can be aborted and restarted later. Another possibility is to autosave the network state after every simulation round, resulting in a complete history of a protocol's states for later inspection. Finally, the tag system can be used to pass an algorithm's results to a subsequent one.

Visualization: Shawn contains two visualization methods. The first one is an interface to the Java front-end "SpyGlass" [BPF+05] that can display both Shawn's simulations and the output of the ESB 430/2 sensor nodes by Scatterweb GmbH (see Section 2.1.1). The second one is a Shawn module that can render the simulation state in various ways, using Cairo[6] as back-end. The module is actually powerful enough to render image sequences. There is a video [FK06a] presenting most of Chapter 3, whose graphics were completely rendered with it.

Routing: Only direct communication is modeled in Shawn's core library (see Section 6.2.2 above). The much more complex issue of routing is addressed in an extension module. It offers an abstract prototype for routing protocols, with arbitrary types of addresses. This allows the user to decide what routing protocol runs underlying an application, without the need to tweak the application's implementation. Protocol implementations for a dummy model and for GeoRouting [WW07] are provided.

Readings: As sensor networks often deal with *sensing*, it is useful to have a common abstraction of sensors. Shawn contains a module that defines a *Reading* class, which is essentially a mapping from $\mathbb{R}^3$ to an arbitrary value type. Applications can reference readings based on user-provided identifiers, and there is a large collection of various types of readings.

Topology: To have a sense of the environment, Shawn has a module that deals with topologies. A topology is basically a reading with boolean value type. The values define which points belong to the network area. There are predefined implementations for 2- and 3-dimensional topologies and for 2D topologies lifted onto a height field. Furthermore, there are generators that sample points from a topology to build a network. Topologies are stored in a pool that can be accessed from the models. Hence, environment-dependent communication is easily implemented.

[6] `http://cairographics.org`

Chapter 7

Conclusion

In this thesis, we presented results from three different areas in wireless sensor networks, namely localization, flows, and simulation.

We introduced a novel type of localization knowledge, where the nodes of a sensor network organize themselves in clusters. First, we dealt with the problem of identifying the boundary of the network without using the positions of the nodes. For that matter, the nodes identify two types of sub-structures: Flowers and augmenting cycles. They maintain cycles in the communication graph and mark nodes that are guaranteed to be on the inside of these cycles, independent of the actual embedding of the nodes. We presented a heuristic by which the nodes can grow the cycles and increase the number of inner nodes, until they span as much of the network as possible. We proved that the resulting geometry description is correct, without the nodes having to compute coordinates for any node. We presented simulative results and comparisons with related algorithms, demonstrating that both the result as well the ability of our algorithm to choose parts of the network for which it can produce feasible results is outstanding.

Then, we turned the knowledge about the network's boundary into a clustering scheme. Utilizing geometric properties from the medial axis of the network area, we constructed two types of clusters: Vertex clusters, which are convex and take the role of "intersections", and tunnel clusters, which have up two adjacent clusters, serving as "streets" resp. "dead-ends". We used a geometric proof technique, where we assume an infinite distribution of the nodes. We described how we obtain a cluster graph, how to enrich it with descriptive cluster properties, and how to use it in applications like routing.

We demonstrated that implementing a form of location knowledge that does not depend on Euclidean coordinates is possible. Generating information about the geometry without access to the node positions is a fascinating topic, especially as obtaining positions is an NP-hard problem in almost all settings. This opens a new field of research and raises many questions. Our work concentrated on a homogeneous WSN, in the sense of devices with identical properties and abilities. Furthermore, we only dealt with high node counts. The case of small, heterogeneous networks is left open, and should be addressed in the future.

In Chapter 5, we introduced a novel network flow problem. We considered finding a maximum flow between two nodes, taking battery constraints and the temporal properties of store-and-forward multi-hop networks into account. The resulting problem was a dynamic flow problem, but the additional constraints make it a new problem for which no directly applicable properties were known. Due to the lack of results on this problem, we presented centralized as well as distributed results. We showed NP-hardness for several cases, and showed that there is an FPTAS that can be implemented in a distributed network.

A tantalizing open problem in this context is the complexity of the fractional 1-ECDF problem itself. Our conjecture is that it is in P. We believe that allowing flow changes only in the first and last n steps does not change the problem. Then, it would be in P, because its dual separation problem can be easily solved similar to Lemma 5.5. Alas, we lack a proof.

Finally, we gave a brief overview over our network simulator Shawn. It is an extremely fast and extensible simulation framework, which we used for all our algorithms. It can easily process networks with up to 500 000 nodes, whereas the competing tools can handle no more than 2 000 without taking unacceptable running times. Shawn is under active development and gaining popularity, so it has a promising future.

Bibliography

[AGY04] James Aspnes, David K. Goldenberg, and Yang Richard Yang. On the computational complexity of sensor network localization. In *Proceedings of the 1st Intl. Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS'04)*, pages 32–44, 2004.

[AKJ05] Nadeem Ahmed, Salil S. Kanhere, and Sanjay Jha. The holes problem in wireless sensor networks: a survey. *ACM SIGMOBILE Mobile Computing and Communications Review*, 9(2):4–18, 2005.

[AKJ06] Nadeem Ahmed, Salil S. Kanhere, and Sanjay Jha. Efficient boundary estimation for practical deployment of mobile sensors in hybrid sensor networks. In *Proceedings of the 3rd IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS'06)*, pages 662–667. IEEE Press, 2006.

[AMO93] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, NJ, 1993.

[AMTW05] Annalingam Anandarajah, Kevin Moore, Andreas Terzis, and I-J. Wang. Sensor networks for landslide detection. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems (SENSYS'05)*, pages 268–269. ACM Press, 2005.

[Bag05] Aline Baggio. Wireless sensor networks in precision agriculture. In *Proceedings of the 2005 ACM Workshop on Real-World Wireless Sensor Networks (REALWSN'05)*, 2005.

[BFN01] Lali Barrière, Pièrre Fraigniaud, and Lata Narayanan. Robust position-based routing in wireless ad hoc networks with unstable transmission ranges. In *Proceedings of the 5th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (Dial-M'01)*, pages 19–27. ACM Press, 2001.

[BGJ05a] Jehoshua Bruck, Jie Gao, and Anxiao (Andrew) Jiang. Localization and routing in sensor networks by local angle information. In *Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Net-*

working and Computing (MOBIHOC'05), pages 181–192. ACM Press, 2005.

[BGJ05b] Jehoshua Bruck, Jie Gao, and Anxiao (Andrew) Jiang. MAP: Medial axis based geometric routing in sensor networks. In *Proceedings of the 11th Annual International Conference on Mobile Computing and Networking (MOBICOM'05)*. ACM Press, 2005.

[BGMS06] Amitabh Basu, Jie Gao, Joseph S.B. Mitchell, and Girishkumar Sabhnani. Distributed localization using noisy distance and angle information. In *Proceedings of the 7th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC'06)*, pages 262–273. ACM Press, 2006.

[BHE06] Marcel Busse, Thomas Haenselmann, and Wolfgang Effelsberg. A comparison of lifetime-efficient forwarding strategies for wireless sensor networks. In *Proceedings of the 3rd ACM International Workshop on Performance Evaluation of Wireless Ad Hoc, Sensor and Ubiquitous Networks (PE-WASUN'05)*, pages 33–40. ACM Press, 2006.

[Bie02] Daniel Bienstock. *Potential Function Methods for Approximately Solving Linear Programming Problems: Theory and Practice*. Kluwer Academic Publishers Group, Boston, MA, 2002.

[BK98] Heinz Breu and David G. Kirkpatrick. Unit disk graph recognition is NP-hard. *Computational Geometry: Theory and Applications*, 9(1-2):3–24, 1998.

[Blu67] Harry Blum. A transformation for extracting new descriptors of shape. In Weiant Wathen-Dunn, editor, *Models for the Perception of Speech and Visual Form*, pages 362–380. MIT Press, 1967.

[BPF+05] Carsten Buschmann, Dennis Pfisterer, Stefan Fischer, Sándor P. Fekete, and Alexander Kröller. Spyglass: A wireless sensor network visualizer. *SIGBED Review*, 2(1):1–6, 2005. Special Issue on the Best of Sensys 2004 Work-in-Progress.

[BRY+04] Maxim A. Batalin, Mohammad Rahimi, Yan Yu, Duo Liu, Aman Kansal, Gaurav S. Sukhatme, William J. Kaiser, Mark Hansen, Gregory J. Pottie, Mani Srivastava, and Deborah Estrin. Call and response: experiments in sampling the environment. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SENSYS'04)*, pages 25–38. ACM Press, 2004.

[CCM97] Hyeong I. Choi, Sung W. Choi, and Hwan P. Moon. Mathematical theory of medial axis transform. *Pacific Journal of Mathematics*, 181(1):57–88, 1997.

[CDB+04] Krishna Kant Chintalapudi, Karthik Dantu, Sandeep Babel, Ramesh Govindan, Gaurav Sukhatme, and John Caffrey. A sensor-actuator network for damage detection in civil structures. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SENSYS'04)*, page 323. ACM Press, 2004.

[CKMS04] Jasmeet Chhabra, Nandakishore Kushalnagar, Benjamin Metzler, and Allen Sampson. Sensor networks in intel fabrication plants. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SENSYS'04)*, page 324. ACM Press, 2004.

[CT04] Jae-Hwan Chang and Leandros Tassiulas. Maximum lifetime routing in wireless sensor networks. *IEEE/ACM Transactions on Networking*, 12(4):609–619, 2004.

[EGW+04] Tolga Eren, David K. Goldenberg, Walter Whiteley, Yang Richard Yang, A. Stephen Morse, Brian D.O. Anderson, and Peter N. Belhumeur. Rigidity, computation, and randomization in network localization. In *Proceedings of the 23nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'04)*, pages 2673–2684, 2004.

[FEdFC02] Ricardo Fabbri, Leandro F. Estrozi, and Luciano da F. Costa. On voronoi diagrams and medial axes. *Mathematical Imaging and Vision*, 17(1):27–40, 2002.

[FF58] Lester R. Ford and Delbert R. Fulkerson. Constructing maximal dynamic flows from static flows. *Operations Research*, 6:419–433, 1958.

[FGG04] Qing Fang, Jie Gao, and Leonidas J. Guibas. Locating and bypassing routing holes in sensor networks. In *Proceedings of the 23nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'04)*, 2004.

[FK06a] Sándor P. Fekete and Alexander Kröller. Geometry-based reasoning for a large sensor network. In *Proceedings of the 22th Annual Symposium on on Computational Geometry (SCG'06)*, pages 475–476, 2006. Video available at http://compgeom.poly.edu/acmvideos/socg06video/index.html.

[FK06b] Stefan Funke and Christian Klein. Hole detection or: how much geometry hides in connectivity? In *Proceedings of the 22th Annual Symposium on on Computational Geometry (SCG'06)*, pages 377–385, 2006.

[FKB+05] Sándor P. Fekete, Alexander Kröller, Carsten Buschmann, Stefan Fischer, and Dennis Pfisterer. Koordinatenfreies Lokationsbewusstsein. *it - Information Technology*, 47:70–78, 2005.

[FKFP07] Sándor P. Fekete, Alexander Kröller, Stefan Fischer, and Dennis Pfisterer. Shawn: The fast, highly customizable sensor network simulator. In *Proceedings of the 4th International Conference on Networked Sensing Systems (INSS 2007)*, page 299, 2007.

[FKKL05] Sándor P. Fekete, Michael Kaufmann, Alexander Kröller, and Katharina A. Lehmann. A new approach for boundary recognition in geometric sensor networks. In *Proceedings of the 17th Canadian Conference on Computational Geometry (CCCG '05)*, pages 82–85, 2005.

[FKP+04] Sándor P. Fekete, Alexander Kröller, Dennis Pfisterer, Stefan Fischer, and Carsten Buschmann. Neighborhood-based topology recognition in sensor networks. In *Proceedings of the 1st Intl. Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS'04)*, volume 3121 of *Lecture Notes in Computer Science*, pages 123–136. Springer-Verlag, 2004.

[Fun05] Stefan Funke. Topological hole detection in wireless sensor networks and its applications. In *Proceedings of the DIALM-POMC Joint Workshop on Foundations of Mobile Computing (DIALM-POMC 2005)*, pages 44–53. ACM Press, 2005.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, NY, USA, 1979.

[GK98] Naveen Garg and Jochen Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS'98)*, page 300. IEEE Computer Society Press, 1998.

[GLS81] Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

[HCL+04] Tian He, Qiuhua Cao, Liqian Luo, Ting Yan, Lin Gu, John A. Stankovic, and Tarek F. Abdelzaher. Electronic tripwires for power-efficient surveillance and target classification. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SENSYS'04)*, pages 315–315. ACM Press, 2004.

[HKS+04] Tian He, Sudha Krishnamurthy, John A. Stankovic, Tarek F. Abdelzaher, Liqian Luo, Radu Stoleru, Ting Yan, Lin Gu, Jonathan Hui, and Bruce Krogh. An energy-efficient surveillance system using wireless sensor networks. In *Proceedings of the 2nd International Conference on Mobile Systems, Applications, and Services (MOBISYS'04)*, 2004.

[Hop95] Bruce E. Hoppe. *Efficient dynamic network flow algorithms*. PhD thesis, Cornell University, 1995.

[JOW+02] Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Li Shiuan Peh, and Daniel Rubenstein. Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with zebranet. *ACM SIGOPS Operating Systems Review*, 36(5):96–107, 2002.

[KAB+05] Lakshman Krishnamurthy, Robert Adler, Phil Buonadonna, Jasmeet Chhabra, Mick Flanigan, Nandakishore Kushalnagar, Lama Nachman, and Mark Yarvis. Design and deployment of industrial sensor networks: experiences from a semiconductor plant and the north sea. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems (SENSYS'05)*, pages 64–75. ACM Press, 2005.

[KDD04] Uwe Kubach, Christian Decker, and Ken Douglas. Collaborative control and coordination of hazardous chemicals. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SENSYS'04)*, page 309. ACM Press, 2004.

[KFPF06] Alexander Kröller, Sándor P. Fekete, Dennis Pfisterer, and Stefan Fischer. Deterministic boundary recognition and topology extraction for large sensor networks. In *Proceedings of the 17th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'06)*, pages 1000–1009, 2006.

[KKP99] Joe M. Kahn, Randy H. Katz, and Kristofer S.J. Pister. Next century challenges: mobile networking for "smart dust". In *Proceedings of the 5th Annual International Conference on Mobile Computing and Networking (MOBICOM'99)*, pages 271–278. ACM Press, 1999.

[KMW04] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. Unit disk graph approximation. In *Proceedings of the DIALM-POMC Joint Workshop on Foundations of Mobile Computing (DIALM-POMC 2004)*, 2004.

[KPB+05] Alexander Kröller, Dennis Pfisterer, Carsten Buschmann, Sándor P. Fekete, and Stefan Fischer. Shawn: A new approach to simulating wireless sensor networks. In *Proceedings of the 3rd Symposium on Design, Analysis, and Simulation of Distributed Systems (DASD'05)*, pages 117–124, 2005.

[KPC+06] Sukun Kim, Shamim Pakzad, David Culler, James Demmel, Gregory Fenves, Steve Glaser, and Martin Turon. Wireless sensor networks for structural health monitoring. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SENSYS'06)*, pages 427–428. ACM Press, 2006.

[Kuh05] Fabian Kuhn. *The Price of Locality: Exploring the Complexity of Distributed Coordination Primitives*. PhD thesis, ETH Zürich, December 2005.

[KW05] Fabian Kuhn and Roger Wattenhofer. Constant-time distributed dominating set approximation. *Distributed Computing*, 17(4):303–310, 2005.

[KWZ03] Fabian Kuhn, Roger Wattenhofer, and Aaron Zollinger. Ad-hoc networks beyond unit disk graphs. In *Proceedings of the DIALM-POMC Joint Workshop on Foundations of Mobile Computing (DIALM-POMC 2003)*, 2003.

[LCD03] Jessica D. Lundquist, Daniel R. Cayan, and Michael D. Dettinger. Meteorology and hydrology in Yosemite national park: A sensor network application. In *Proceedings of the 2nd International Symposium on Information Processing in Sensor Networks (IPSN'03)*, 2003.

[LR03] Koen Langendoen and Niels Reijers. Distributed localization in wireless sensor networks: A quantitative comparison. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 43(4):499–518, 2003.

[Mit91] Joseph S.B. Mitchell. A new algorithm for shortest paths among obstacles in the plane. *Annals of Mathematics and Artificial Intelligence*, 3:83–105, 1991.

[ML06] Ritesh Madan and Sanjay Lall. Distributed algorithms for maximum lifetime routing in wireless sensor networks. *IEEE Transactions on Wireless Communications*, 5(8):2185–2193, 2006.

[MLL05] Ritesh Madan, Zhi-Quan Luo, and Sanjay Lall. A distributed algorithm with linear convergence for maximum lifetime routing in wireless networks. In *Proceedings of the 43rd Annual Allerton Conference on Communication, Control, and Computing*, pages 896–905, 2005.

[MOWW04] Thomas Moscibroda, Regina O'Dell, Mirjam Wattenhofer, and Roger Wattenhofer. Virtual coordinates for ad hoc and sensor networks. In *Proceedings of the DIALM-POMC Joint Workshop on Foundations of Mobile Computing (DIALM-POMC 2004)*, 2004.

[MPR+05] Kirk Martinez, Paritosh Padhy, Alistair Riddoch, Royan Ong, and Jane Hart. Glacial environment monitoring using sensor networks. In *Proceedings of the 2005 ACM Workshop on Real-World Wireless Sensor Networks (REALWSN'05)*, 2005.

[MPS+02] Alan Mainwaring, Joseph Polastre, Robert Szewczyk, David Culler, and John Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02)*, 2002.

[MS04] Fernando Martincic and Loren Schwiebert. Distributed perimeter detection in wireless sensor networks. Technical report, Wayne State University, 2004. Report ID WSU-CSC-NEWS/03-TR03.

[NM03] Robert Nowak and Urbashi Mitra. Boundary estimation in sensor networks: Theory and methods. In *Proceedings of the 2nd International Symposium on Information Processing in Sensor Networks (IPSN'03)*, volume 3121 of *Lecture Notes in Computer Science*, pages 80–95. Springer-Verlag, 2003.

[NN01] Dragos Niculescu and Badri Nath. Ad-hoc positioning system (APS). In *Proceedings of the 44th IEEE Global Telecommunications Conference (GLOBECOM'01)*, pages 2926–2931, 2001.

[Pap94] Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[Pel00] David Peleg. *Distributed computing: a locality-sensitive approach*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[Pfi07] Dennis Pfisterer. *Comprehensive Development Support for Wireless Sensor Networks*. PhD thesis, University of Lübeck, 2007.

[Sch86] Alexander Schrijver. *Theory of linear and integer programming*. John Wiley and Sons, New York, NY, USA, 1986.

[SCL+05] Victor Shnayder, Bor-Rong Chen, Konrad Lorincz, Thaddeus R. F. Fulford Jones, and Matt Welsh. Sensor networks for medical care. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems (SENSYS'05)*, page 314. ACM Press, 2005.

[SL04] Arvind Sankar and Zhen Liu. Maximum lifetime routing in wireless ad-hoc networks. In *Proceedings of the 23nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'04)*, pages 1089–1097, 2004.

[SMAL+04] Gyula Simon, Miklós Maróti, Ákos Lédeczi, György Balogh, Branislav Kusy, András Nádas, Gábor Pap, János Sallai, and Ken Frampton. Sensor network-based countersniper system. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SENSYS'04)*, pages 1–12. ACM Press, 2004.

[SMP+04] Robert Szewczyk, Alan Mainwaring, Joseph Polastre, John Anderson, and David Culler. An analysis of a large scale habitat monitoring application. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SENSYS'04)*, pages 214–226. ACM Press, 2004.

[SPS02] Andreas Savvides, Heemin Park, and Mani B. Srivastava. The bits and flops of the N-hop multilateration primitive for node localization problems. In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02)*, pages 112–121, 2002.

[SPW96] Evan C. Sherbrooke, Nicholas M. Patrikalakis, and Franz-Erich Wolter. Differential and topological properties of medial axis transforms. *Graphical Models and Image Processing*, 58(6):574–592, 1996.

[SRL02] Chris Savarese, Jan M. Rabaey, and Koen Langendoen. Robust positioning algorithms for distributed ad-hoc wireless sensor networks. In *USENIX Technical Annual Conference'02*, pages 317–327. USENIX Association, 2002.

[STM+05] A. Sheth, K. Tejaswi, P. Mehta, C. Parekh, R. Bansal, S. Merchant, T. Singh, U. B. Desai, C. A. Thekkath, and K. Toyama. SenSlide: a sensor network based landslide prediction aystem. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems (SENSYS'05)*, pages 280–281. ACM Press, 2005.

[TPS+05] Gilman Tolle, Joseph Polastre, Robert Szewczyk, David Culler, Neil Turner, Kevin Tu, Stephen Burgess, Todd Dawson, Phil Buonadonna, David Gay, and Wei Hong. A macroscope in the redwoods. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems (SENSYS'05)*, pages 51–63. ACM Press, 2005.

[VKR+05] Iuliu Vasilescu, Keith Kotay, Daniela Rus, Matthew Dunbabin, and Peter Corke. Data collection, storage, and retrieval with an underwater sensor network. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems (SENSYS'05)*, pages 154–165. ACM Press, 2005.

[WASW06] Geoffrey Werner-Allen, Patrick Swieskowski, and Matt Welsh. Real-time volcanic earthquake localization. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SENSYS'06)*, pages 357–358. ACM Press, 2006.

[WGM06] Yue Wang, Jie Gao, and Joseph S.B. Mitchell. Boundary recognition in sensor networks by topological methods. In *Proceedings of the 12th Annual International Conference on Mobile Computing and Networking (MOBICOM'06)*, pages 122–133. ACM Press, 2006.

[WW07] Dorothea Wagner and Roger Wattenhofer, editors. *Algorithms for Sensor and Ad Hoc Networks*, volume 4621 of *Lecture Notes in Computer Science*. Springer-Verlag, 2007.

[XRC+04] Ning Xu, Sumit Rangwala, Krishna Kant Chintalapudi, Deepak Ganesan, Alan Broad, Ramesh Govindan, and Deborah Estrin. A wireless sensor network for structural monitoring. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SENSYS'04)*, pages 13–24. ACM Press, 2004.

[ZG04] Feng Zhao and Leonidas J. Guibas. *Wireless Sensor Networks: An Information Processing Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[ZHKS04] Gang Zhou, Tian He, Sudha Krishnamurthy, and John A. Stankovic. Impact of radio irregularity on wireless sensor networks. In *Proceedings of the 2nd International Conference on Mobile Systems, Applications, and Services (MOBISYS'04)*, pages 125–138. ACM Press, 2004.

[ZS03] Gil Zussman and Adrian Segall. Energy efficient routing in ad hoc disaster recovery networks. In *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'03)*, pages 682–691, 2003.

[ZSG07] Xianjin Zhu, Rik Sarkar, and Jie Gao. Shape segmentation and applications in sensor networks. In *Proceedings of the 26th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'07)*, pages 1838–1846, 2007.

[ZZF06] Chi Zhang, Yanchao Zhang, and Yuguang Fang. Localized coverage boundary detection for wireless sensor networks. In *Proceedings of the 3rd international conference on Quality of service in heterogeneous wired/wireless networks (QShine '06)*, page 12. ACM Press, 2006.

Index

Zusammenfassung

Die vorliegende Arbeit beschäftigt sich mit algorithmischen und geometrischen Fragestellungen in Sensornetzwerken. In den 1990er Jahren entwickelte sich die Idee von unsichtbar kleinen, intelligenten und vernetzten Partikeln von futuristischer Träumerei hin zu greifbar nahen technischen Möglichkeiten. Das SmartDust-Projekt [KKP99] der UC Berkeley beschrieb detailliert, wie sogenannte „Motes“, winzige Computer in der Größe eines Kubikmillimeters, zu konstruieren seien. Bereits heute existieren Geräte mit einer Vielzahl von Anwendungen, und die fortschreitende Miniaturisierung und der mit steigender Produktion einhergehende Preisverfall werden in den nächsten Jahren große Sensornetzwerke ermöglichen.

Klassische Algorithmen verwenden Berechnungsmodelle, in denen ein einzelner Prozessor sequenziell Anweisungen abarbeitet, und dabei Zugriff auf die vollständigen Daten eines Problems hat. In Sensornetzwerken sind derartige Modelle bedeutungslos. Es werden verteilte Algorithmen benötigt, bei denen die einzelnen Prozessoren miteinander Daten austauschen, Berechnungen durchführen und gemeinsam eine Aufgabe bewältigen, zu der einzelne Geräte niemals in der Lage wären. Weitere Herausforderungen bestehen darin, dass die Prozessoren sehr einfach, die Speicherausstattung klein, und die Energieversorgung gering ist. Zusätzlich kann nicht jedes Gerät mit jedem anderen beliebig Daten austauschen, sondern nur mit solchen, die es mittels seiner Funkschnittstelle erreichen kann. Dadurch spielt die geometrische Verteilung eine entscheidende Rolle.

Diese Arbeit umfasst neben Einführung und Überblick in das Thema sowie einer Voruntersuchung drei wesentliche Teile:

Der erste Teil schlägt sich in Kapitel 3 und 4 nieder. Darin untersuchen wir, wie die Knoten des Netzwerks ihre Position bestimmen und ausnutzen können, ohne auf euklidische Koordinaten zurückgreifen zu müssen. Das Problem, die Knotenpositionen in einem zweidimensional eingebetteten Graphen zu bestimmen, ist in nahezu allen für Sensornetzwerke relevanten Varianten NP-schwer, und die existierenden Berechnungsverfahren haben entweder gar keine oder nur eine völlig unzureichende beweisbare Güte.

Wir stellen daher ein alternatives Verfahren vor. Es wird aufgezeigt, wie das Sensornetzwerk einen Bereich identifizieren kann und den Beweis führt, dass bestimmte Knoten am Rand dieses Bereiches liegen müssen. Das wesentliche Unterscheidungsmerkmal gegenüber allen vergleichbaren Heuristiken besteht im Beweis der Korrektheit: Wir kennen kein Verfahren, das den Rand des Netzwerks korrekt

bestimmt, insbesondere da eine exakte Definition des Randbegriffs nicht existiert. Unser Algorithmus hingegen entscheidet selbständig, wo er einen vernünftig definierbaren Rand erkennen kann, und markiert sowohl diesen als auch den zugehörigen Bereich.

Danach zeigen wir, wie der erkannte Rand zur Erzeugung eines Positionsbegriffes genutzt werden kann. Wir definieren für den rein geometrischen Fall eine Zerlegung des Gebiets, in dem die Sensorknoten verteilt sind. Dabei entstehen zwei Arten von Gebieten: Die einen sind konvex und haben mindestens drei Kontakte zu Nachbargebieten, sie bilden also eine Art Kreuzung. Die anderen haben immer exakt zwei Kontakte, stellen also Verbindungsstraßen dar. Eine derartige Zerlegung existiert immer. Der durch die Kontakte definierte „Clustergraph“ beschreibt die topologische Struktur des zugrundeliegenden Gebietes sehr genau. Wir stellen verteilte Heuristiken vor, durch die jeder Knoten feststellt, in welchem Gebiet der Zerlegung er sich befindet. Auch diese Verfahren greifen auf keinerlei koordinatenkodierte Informationen zurück.

Das sich daraus ergebende Positionsbewusstsein ist für einige Anwendungen dem klassischen Ansatz deutlich überlegen. Wir beschreiben, wie Routingverfahren dadurch verbessert werden können.

Der zweite Teil, Kapitel 5, widmet sich einer Adaption eines Flussproblems für Sensornetzwerke. Wir gehen davon aus, dass jeder Sensorknoten eine eigenständige, nicht wiederaufladbare Batterie hat. Das Senden und Empfangen von Nachrichten ist energieaufwändig. Des Weiteren kann ein entladener Knoten nicht mehr an den Berechnungen im Netzwerk teilnehmen. Wenn ein Knoten Daten zu einem weit entfernten Empfänger übertragen will, muss er sich einiger Zwischenknoten bedienen. Dabei wird die Übertragung verlangsamt, da in jedem Schritt ein Datenpaket empfangen, überprüft und weitergeschickt werden muss. Daher betrachten wir das „Energy-Constrained Dynamic Flow Problem“: Gegeben ist ein Sensornetzwerk mit ausgezeichneter Quelle und Senke, sowie Kantenkapazitäten, Kantenlaufzeiten, Batterieladestände und ein Zeithorizont. Gesucht ist ein Fluss, der die innerhalb des Horizonts die Senke erreichende Datenmenge maximiert. Dabei handelt es sich um eine Variante bekannter dynamischer Flussprobleme, die in dieser Form aber noch nicht analysiert wurde.

Aufgrund fehlender Vorarbeiten haben wir dieses Problem vollständig, also insbesondere auch im klassischen, zentralisierten Modell untersucht. Wir beweisen, dass das Problem NP-schwer ist, und vermutlich nicht einmal ein Kodierungsschema für Lösungen besitzt, das polynomiell beschränkt ist. Wir entwickeln zwei Approximationsschemata, ein zentralisiertes und ein verteiltes, für den für Sensornetze einzig relevanten Fall, in dem alle Kantenlaufzeiten 1 betragen.

Der dritte Teil (Kapitel 6) stellt kurz den Netzwerksimulator Shawn vor. Algorithmen und Protokolle für Sensornetze werden oft nur simuliert, da die Knoten

derzeit noch sehr teuer sind und das Aufspielen neuer Software im Allgemeinen extrem aufwändig ist.

Unglücklicherweise existierten vormals ausschließlich Simulationsumgebungen, die sich auf die möglichst akkurate Emulation sehr einfacher Protokolle für kleine Sensornetze beschränkten. Ein Programm, mit dem sich Algorithmen auch auf sehr großen Netzwerken testen lässt, existierte nicht.

Daher ist im Rahmen dieser und anderer Arbeiten Shawn entstanden. Es handelt sich dabei um ein Softwarepaket, das sowohl verteilte als auch zentralisierte Algorithmen ausführen kann. Der Benutzer kann zwischen verschiedenen geometrischen Kommunikationsmodellen wählen und das Speichermodell für den daraus resultierenden Graphen an den verfügbaren Hauptspeicher sowie an Simulationsparameter wie eventuell mögliche Mobilität der Knoten anpassen.

Im direkten Vergleich mit dem bekanntesten Simulator, Ns-2, ergab sich, dass Shawn Protokolle auf 1000 Knoten in unter fünf Minuten simuliert, während Ns-2 über 25 Stunden rechnete und damit an seine Leistungsgrenze kam. Für die Simulation unserer Algorithmen aus dem ersten Teil kamen Netzwerke mit ca. 30 000 Knoten zum Einsatz, und für andere Arbeiten konnten auch schon Simulationen mit 500 000 Knoten ohne Probleme durchgeführt werden.